Cape Peninsula
University of Technology

Dynamic Superscalar Grid for Technical Debt Reduction

by

Rudi Killian 193007959

Thesis submitted in fulfilment of the requirements for the degree

Master of Technology: Information Technology

in the Faculty of Informatics and Design

at the Cape Peninsula University of Technology

Supervisor: Dr. Boniface Kabaso
Co-supervisor: Amlan Mukherjee

Cape Town
May 2018

DECLARATION

I, Rudi Killian, declare that the contents of this dissertation/thesis represent my own unaided work, and that the dissertation/thesis has not previously been submitted for academic examination towards any qualification. Furthermore, it represents my own opinions and not necessarily those of the Cape Peninsula University of Technology.

Signed **Date**

ABSTRACT

Organizations and the private individual, look to technology advancements to increase their ability to make informed decisions. The motivation for technology adoption by entities sprouting from an innate need for value generation. The technology currently heralded as the future platform to facilitate value addition, is popularly termed cloud computing. The move to cloud computing however, may conceivably increase the obsolescence cycle for currently retained Information Technology (IT) assets. The term obsolescence, applied as the inability to repurpose or scale an information system resource for needed functionality. The incapacity to reconfigure, grow or shrink an IT asset, be it hardware or software is a well-known narrative of technical debt. The notion of emergent technical debt realities is professed to be all but inevitable when informed by Moore's Law, as technology must inexorably advance. Of more imminent concern however are that major accelerating factors of technical debt are deemed as non-holistic conceptualization and design conventions. Should management of IT assets fail to address technical debt continually, the technology platform would predictably require replacement. The unrealized value, functional and fiscal loss, together with the resultant e-waste generated by technical debt is meaningfully unattractive.

Historically, the cloud milieu had evolved from the grid and clustering paradigms which allowed for information sourcing across multiple and often dispersed computing platforms. The parallel operations in distributed computing environments are inherently value adding, as enhanced effective use of resources and efficiency in data handling may be achieved. The predominant information processing solutions that implement parallel operations in distributed environments are abstracted constructs, styled as High Performance Computing (HPC) or High Throughput Computing (HTC). Regardless of the underlying distributed environment, the archetypes of HPC and HTC differ radically in standard implementation. The foremost contrasting factors of parallelism granularity, failover and locality in data handling have recently been the subject of greater academic discourse towards possible fusion of the two technologies.

In this research paper, we uncover probable platforms of future technical debt and subsequently recommend redeployment alternatives. The suggested alternatives take the form of scalable grids, which should provide alignment with the contemporary nature of individual information processing needs. The potential of grids, as efficient and effective information sourcing solutions across geographically dispersed heterogeneous systems are envisioned to reduce or delay aspects of technical debt. As part of an experimental investigation to test plausibility of concepts, artefacts are designed to generically implement HPC and HTC. The design features exposed by the experimental artefacts, could provide insights towards amalgamation of HPC and HTC.

Keywords:
Design Science Research, Grid Computing, Heterogeneous Platforms, High Performance Computing, High Throughput Computing, Technical Debt

ACKNOWLEDGEMENTS

I wish to thank:

- **Boniface Kabaso - for his patience, enthusiasm & unyielding support**
- **Amlan Mukherjee – for championing & defending technical research papers**
- **Wilhelm Rothman – for his constructive critique**
- **Varsity College CPT – for providing computer lab infrastructure**
- **The MASM32 forum – for beta testing various artefact constructs**
- **Intel Corporation – Steve Caroompas & Ronald Benson for providing access to Parallel Studio XE Cluster Edition for Windows**
- **Published IS scholars – on who’s giant shoulders we may stand and hope to be counted worthy**

DEDICATION

I would like to dedicate this work to my past and present mentors, teachers and lecturers; without whom my academic ambitions would not have endured. Enormous appreciation and devotion is expressed to my wife and son for providing me the drive and allowing me the time, to complete the paper.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF CODELETS

GLOSSARY

Acronyms/Abbreviations	Definition/Explanation
API	Application Programmable Interface
APIC	Advanced Programmable Interrupt Controller
AVX	Advanced Vector Extensions
CPU	Central Processing Unit
DFS	Distributed File System
DMA	Direct Memory Access
DSR	Design Science Research
FEDS	Framework for Evaluation in Design Science Research
FMA	Fused Multiply and Addition
GB	Gigabyte
GUID	Globally Unique Identifier
HPC	High Performance Computing
HTC	High Throughput Computing
I/O	Input / Output
ICT	Information & Communications Technologies
IDE	Integrated Development Environment
IOCP	I/O Completion Ports
IP	Internet Protocol
IS	Information System
IT	Information Technology
iWARP	Internet Wide Area RDMA Protocol
KB	Kilobytes
LAN	Local Area Network
LLC	Last Level Cache
MB	Megabyte
MIMD	Multiple Instruction, Multiple Data
MMX	Matrix Multiplication Extensions
MPI	Message Passing Interface
MPKI	Cache Misses per 1000 Instructions
MTU	Maximum Transmission Unit
NIC	Network Interface Card / Adapter
NUMA	Non-Uniform Memory Access
OS	Operating System
PMU	Performance Monitoring Unit
RAM	Random Access Memory
RDMA	Remote Direct Memory Access
RoCE	RDMA over Converged Ethernet
RPC	Remote Procedure Call
SAN	System Area Network
SIMD	Single Instruction, Multiple Data
SISD	Single Instruction, Single Data
SSE	Streaming SIMD Extensions
TB	Terabyte
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VM	Virtual Machine
WAN	Wide Area Network
WINE	Wine Is Not an Emulator
x64	Compatible to the 64-bit Intel instruction set
x86	Compatible to the 32-bit Intel instruction set

CHAPTER ONE
INTRODUCTION

1.1 Background to Research Problem

Organizations are aware that business intelligence is key to competitive advantage (Ward & Peppard, 2002; Schiesser, 2010; Bidgoli, 2016). The inquiries concerning business intelligence gathering exercises within the contemporary setting, notes scholarly agreement, in that organisations are increasing reliant on the ability to leverage data (Chen, et al., 2012; Dhar & Mazumder, 2014; Hashem, et al., 2015). How such business intelligence from data could be facilitated, may potentially be of critical interest to organisational decision makers.

The mechanisms of business intelligence normally regard the utilization of tools and techniques to transform organizational data into useful information. However, the control, processing and analysis of these prospective data assets may require fresh doctrines in planning as well as implementation. Importantly awareness is required that traditional techniques for data manipulation and analysis, are known to be ineffectual in dealing with the realities of modern day organisational data landscapes (Katal, et al., 2013; Hashem, et al., 2015). When working with data assets from which potential business intelligence might be gleaned, several native difficulties could be uncovered. Cognizance needs to be taken of the internal organizational capabilities, before designing platforms for modern data analysis and demanding workloads, as such capabilities may easily be exceeded (Hashem, et al., 2015). The size and properties of data are at the root of the problem. The major challenges uniformly originating from the growth rate and large varying types of data obtainable. In lieu of solutions towards effective business intelligence generation within an unconventional data or *big data* era, acquaintance with technologies that have the potential to mitigate obstacles in creation are needed. A technology that could produce or enhance capabilities on demand would arguably have direct fit with the organisational capacity problem.

The term *grid computing* was first devised in the 1990's to describe a paradigm by which on demand computing power could be obtained (Foster, et al., 2008). As demand dictates, multiple processing resources combine logically as a singular managed object. The resources and processing capacity of the effectively assembled grid object, may then easily surpass the capabilities of any individual or cooperative constituents. The activity of constructing the managed grid resource is frequently termed as *clustering* or creating a *distributed computing* platform (Foster, et al., 2008; Deelman, 2010). Other appealing benefits obtained from the grid paradigm, are noted to include: better resource utilization, data federation and centralization of control.

The modern manifestation of on demand and distributed computing, is popularly known as *cloud computing*. Indeed, the technological base of current cloud environments are purposefully modelled after grid, distributed and clustered computing platforms (Foster, et al., 2008; Youseff, et al., 2008; Deelman, 2010; Reyes-Ortiz, et al., 2015). Given sufficient scale (i.e., the major semantic distinction between clouds and grids), on demand computing power is universally associated with the existing understanding around the topic of cloud infrastructure formation. The dynamic and enabling capacity generation, as facilitated by grid computing can subsequently also be mimicked by use of cloud computing.

The organisational Information Technology (IT) function and private individuals alike, are increasingly adopting cloud based technological mechanisms to for-fill and augment their mandate (Chen, et al., 2012; Hashem, et al., 2015). Conceivably the IT strategy in an attempt to align with business, would therefore also be looking to popularized cloud technologies as an enabler of business intelligence. The adoption of cloud technologies has inopportunely engendered the creation of a veritable terminology zoo to describe any discreet organisational data endeavours. The frequency of such cloud jargon may serve only to undermine focus and add complexity to the business intelligence building issue. Of interest rather, would be the extensive dialogue of benefits gained from the move to cloud and the use of platform aided data technologies. Foremost of these value propositions are:

- Exploiting economies of scale (Foster, et al., 2008)
- Enhanced fiscal flexibility, based on per unit purchase choices (Foster, et al., 2008)
- High speed processing, produces insights more rapidly (Dobre & Xhafa, 2014)
- High volume of data throughput, produces higher accuracy in analysis (Chang & Wills, 2015)
- Capacity building to deal with exponential growth and storage of data (Gupta, 2015)

The benefits of the cloud computing also transcend business, to include academic research (Deelman, 2010; Masud, et al., 2012; Dobre & Xhafa, 2014; Reyes-Ortiz, et al., 2015; Leetaru, 2016). The same benefits targeted by business, would therefore have equivalency and import for academia. Assuredly, the prevalent trends of incentivized discounts, research grants and unprecedented ease of access to large data sets would appeal to researchers. Inferences of apprehension regarding the move to cloud may however be noted in the research literature. The peer-reviewed concerns predominating around the adoption of cloud technology, before wide-ranging maturity and internal efficiencies of intended functionality are achieved (Dhar & Mazumder, 2014; Grolinger, et al., 2014; Hashem, et al., 2015; Botta, et al., 2016).

The maturity, effectiveness and efficiency theme is arguably not new within the field of IS research, but may require contextual scrutiny. The IT management discipline, per illustration, uses indicators of maturity, effectiveness and efficiency to rectify or address gaps in IT and business alignment (Ward & Peppard, 2002; Schiesser, 2010; Bidgoli, 2016). The subject of cloud adoption trends and their impacts, given these indicators, suggestively imparts the importance of alignment before adoption. The Gartner research group predicts that the majority of IT spend by the year 2020 will be on cloud and related services (Gartner Inc., 2016). Of looming concern would be the non-aligned IT function scenario, where business demands the move to cloud or eliminates the cloud purchase choice.

Even when disregarding IT alignment, deductive reasoning suggests that tangible consequences must somehow manifest should future ICT budgets be diverted towards the cloud. In the immediate aftermath of cloud adoption, the reconfiguration of existing hardware and software architectures would certainly seem inevitable. Structurally not all IT resources might find redeployment in the cloud technology environment, which could result in *technical debt* (Martini, et al., 2014). The term technical debt describes the inability of IT resources to be grown, shrunk or reconfigured, consequently resulting in possible organizational dysfunction. The failure of a resource to be reconfigured or scaled could potentially undermine the business processes being supported. The cause of technical debt is provided as being some form of loss, due to non-holistic decisions made in support of short-term goals, with disregard for long-term outcomes (Martini, et al., 2014). Lack of plans that address common cloud technology pitfalls when juxtaposed with existing resource allocation may lead to such potential loss. The existing computer applications and underlying infrastructure would need either to be reconfigured or made obsolete to facilitate cloud migration. Innovative solutions would be required to address the likelihood of abrupt obsolescence.

The cloud model in the setting of data analytics requires scaling of computing infrastructure, which is multi-node and typically multi-processor. In contemporary data analysis, a given problem domain is frequently subdivided between network nodes, which in turn independently or cooperatively process instructions and data (White, 2012; Dobre & Xhafa, 2014; Doulkeridis & Nørvåg, 2014). Once individual node processing is complete, the result may be unified centrally, producing the information base for the decision-making process. On examination, two distinct and divergent grid application models are therefore classifiable (Foster, et al., 2008; Dobre & Xhafa, 2014): *High Performance Computing* (HPC) and *High Throughput Computing* (HTC).

Both HPC and HTC models have inherent value in application, though their approach to computation and data storage differ radically. HPC provides for tightly coupled parallel runs per computation node, in scenarios where data and instructions have been localized (Deelman, 2010; Dobre & Xhafa, 2014; Reyes-Ortiz, et al., 2015). The HTC application model could potentially support both tightly and loosely coupled parallel runs per computation node, in scenarios where data and/or instructions are externally sourced (Foster, et al., 2008; Youseff, et al., 2008; Deelman, 2010; Shvachko, et al., 2010; Montero, et al., 2011; White, 2012; Reyes-Ortiz, et al., 2015). The word *parallel(ism)* provisionally referring to either the computer processor or networked node-based interaction capabilities of an infrastructure platform. The HPC model derives its efficiencies by attempting balanced targeting of parallelism, per computation node, at processor level. To gain efficiencies, the model is therefore cognizant of higher orders of parallelism at the node level. The HTC model contrasts by delivering better data management capabilities amongst participant nodes. In processor-based parallelism, a single node may have multiple physical processors. Each physical processor package or die, may be further deconstructed to reveal central processing units (CPU's) or cores. A processor package may have multiple cores that independently or cooperatively compute towards a given solution (Intel Corporation, 2016). Modern CPU's additionally contain parallelism at the data and instruction level, termed *vector instructions* they gain processing speed-ups by executing a single instruction that simultaneously operates on multiple data streams per clock-cycle. Vector parallelism and facilitating instruction sets, when used in conjunction with multi-core CPU architectures, are known to significantly increase processing throughput (Akhter & Roberts, 2006; Jeong & Lee, 2012; Sabharwal, et al., 2013; Reyes-Ortiz, et al., 2015; Saxena, et al., 2016). It ought to be evident that technologies that provide for efficient data processing, therefore be designed with mindful and inherent parallelism beyond the multi-node setting. Importantly, parallelism must be reflected in the design of an effective and thus generalizable HPC or HTC solution framework, if any reasonable uptake of such technology is to be expected (Reyes-Ortiz, et al., 2015). The effects of ignoring inherent parallelism could be the underutilization of processing and data resources, which could potentially weaken the value assertion of data analytics. Literature has long shown that data analytics technologies are missing the opportunity to holistically incorporate value-adding features. The remedial recommendations in topical research, overwhelmingly suggests frameworks that could ultimately lead to hybridized versions of HPC and HTC (Alverson, et al., 1992; Montero, et al., 2011; Chang, et al., 2014; Dobre & Xhafa, 2014; Doulkeridis & Nørvåg, 2014; Grolinger, et al., 2014; Zhao, et al., 2014; Mantripragada, et al., 2015; Nelson, et al., 2015; Reyes-Ortiz, et al., 2015; Saxena, et al., 2016).

An additional hindrance to value add, comes from the fact that software developers predominant in industry, are mostly incapable of programming for parallelism (Darlington, et al., 1993; Berlin, et al., 2004; Lee, 2006; Herlihy, 2010; Newburn, et al., 2011; Breivold & Crnkovic, 2014; Saxena, et al., 2016). Programming skillsets that disregard parallelism would consequently negate possible benefits as originally envisioned in the endeavour. To alleviate the shortcomings in skillset, milieu solutions abstract programming languages via application programmable interface, additional libraries or run-time script interpreters. The solution then becomes reliant on compilation interfaces and background services that add additional processing overhead. Compounding the problem is that it might not be reasonably expected for software developers, skilled in parallelism or not, to target a single binary executable for heterogeneous hardware platforms (Berlin, et al., 2004; Newburn, et al., 2011). Developing for homogeneous platforms, would limit future agility and ultimately benefit potential. The selected homogenous platform would dictate the development, testing, scope, operational and maintenance deployment environment of a utilitarian solution (Satzinger, et al., 2012). Current solutions then also further abstract the hardware and storage platform of the network operating system, to facilitate heterogeneous platforms.

The potential to make informed decisions from structured and unstructured data with unprecedented speed is assuredly desirable for business and the scientific researcher alike. The information sourcing and processing requirements necessary for decision making, in the reflected context of the cloud and big data value proposition, have however several obstacles to realization (Deelman, 2010; Kraska, 2013; Dhar & Mazumder, 2014; Grolinger, et al. 2014, Hashem, et al., 2015). During a comprehensive literature review of cloud computing, the open research problems that effect cloud adoption was identified as (Hashem, et al., 2015): regulatory governance concerns, legal issues, scalability, privacy, availability, data quality, data heterogeneity; data integrity and data transformation. Of notice are the problems around *scalability*, *availability* and *data heterogeneity*, which functionally have bearing on data analytics and HPC or HTC. Scalability speaks to the ability to grow or shrink the data storage and infrastructure, as demand or need requires (Schiesser, 2010; Martin, et al., 2014; Hashem, et al., 2015). Availability, defined as having resources available to authorized entities at the time of demand (Schiesser, 2010; Ciampa, 2015). Current computational infrastructures find it problematic to scale data storage and platform, whilst also maintaining availability (Deelman, 2010; Martin, et al., 2014; Hashem, et al., 2015). The mitigations of fault tolerance and various forms of redundancy have been the mainstay solutions to the availability issue in the past. However, when coupling availability with scalability, non-trivial outcomes may be assured.

The heterogeneity of data refers to the characteristics of data begotten from multiple sources. The characteristics of heterogeneous data would present as having different type and size, as well as data quality attributes (Che, et al., 2013; Hashem, et al., 2015). What is more, heterogeneous data has inconsistency in representation. The operationalization of activities that function on heterogeneous data would notably be a technical challenge. Feasibly however, programmable solutions that are informed by the application purpose could be found for specific heterogeneous data environments.

1.2 Research Problem

The move to cloud-based computing could demand structural IT environmental changes leading to substantial amounts of technical debt. The future value proposition of data management and analytical solutions might be undermined by inherent inefficiencies of abstraction due to compensating factors. The existing design solutions that incorporate both high performance computing and high throughput computing are not necessarily generalizable.

1.3 Research Aim

The research aim is to create a software artefact, which could provide insights into the design environment of a generalizable grid implementation. The artefact should amalgamate high performance computing and high throughput computing, whilst addressing future technical debt.

1.4 Research Question

Could innovative application design, facilitate the repurposing of technical debt as grid implementations?

1.5 Research Sub-Questions

1. What hardware and software architectures will constitute the most probable sources of technical debt?
2. What hardware information is required to target grid implementations programmatically?
3. How may hardware and network platforms be enumerated for the grid?
4. What is the predominate functionality exposed by current grid technologies?
5. What enhanced functionality is exposed by the designed artefact?

1.6 Objectives

1. Ascertain the contemporary architectures at risk of technical debt accretion
2. Determine programmable ingredients that would facilitate diverse hardware integration into potential grid platforms
3. Establish a base line of current utility and extents as begotten from existing grid solutions
4. Measure a candidate design artefact's utility and extents against the baseline, highlighting efficacy at technical debt reduction

1.7 Envisioned Contribution

The study aims to contribute theoretical and applied understanding, by designing an innovative software artefact to be used in the grid environment. Design outcomes are proposed to facilitate theory enhancements for the body of knowledge. The artefact itself should create opportunity or alternative for entities facing technical debt issues.

1.8 Research Methodology

The research initial ontological stance is *objectivism* but would not exclude an evolutionary prospect of *pragmatism* over the research timeline.

The term objectivism means that there is an independent reality that may be understood by investigating the laws that govern it (Neuman, 2011; Bryman, et al., 2014). Equally pragmatic values are not discarded as research investigations may lead to new insights within the workings of the perceived reality, which when practically applied, may change the understanding of the phenomenon under investigation (Neuman, 2011; Bryman, et al., 2014). The research epistemology is principally identified as *positivism*. Yet the philosophically purest interpretation of positivism is not expected to be adhered to as software artefact building and evaluation activities, are not synonymous with the epistemology.

A *reductionist* approach, which investigates the smaller constituent components of larger phenomenon is anticipated. Only deductive factual observation and quantifiable truths are of interest. The validity of research findings made are paramount in adding academic value, therefore quantitative data would be collected and analysed.

The research paper's focus is on unravelling an impending problem. An expectantly reproducible exploration, drawing from contemporary theory across multivariate research domains could speculatively provide new understanding. For these reasons the Design Science Research (DSR) paradigm, subordinate strategies, evaluation criteria and methods are implemented.

1.9 Thesis Organisation

The remaining balance of the research paper is ordered as follows:

Chapter 2, at first provides inclusion rational and grounding guidelines for survey literature selection. The subsequent narrative underpins and explores the technical debt phenomenon as an outflow of cloud adoption. To delay the burden of technical debt and e-waste, the repurposing of infrastructure, hardware and software assets are suggested. Accordingly, the research investigation initially identifies probable future technical debt platforms. The identified platforms, are to be the subject of scalable HPC and HTC grid solutions as alternative to potential obsolescence. The nature of parallelism as well as heterogeneity, within HPC and HTC is found to be problematic in literature. Successive research inquiries, delve deeper into the inherent parallelism and heterogeneity of computing platforms. During the course of such inquiries, concerted efforts are made to consistently provide relevant fit for HPC and HTC environs. The understanding generated, should clarify difficulties and suggest ingredients that facilitate an effective and efficient collaborative computing platform. In brief, an atomic examination of a hypothetical HPC and HTC grid node is undertaken. Starting with the application’s interactions with the operating system, a holistic case for reduced abstraction may be conceived. Scrutinizing the hardware, software, network and environmental characteristics of a potential grid participant, could provide insights towards how improved computational efficiency and effectiveness may be achieved.

In **Chapter 3**, the Design Science Research paradigm, its subordinate strategies, processes and methods, together with artefact evaluation are discussed in detail. The quantitative data collection methods, metric characteristics and analysis thereof are explained.

Chapter 4 describes the experimental artefact build environment, as well as the modular and iterative design endeavour. The design elements and constructs developed, are independently compared with similar technology, theories or schemes.

In **Chapter 5**, the design artefact is experimentally instanced and then rigidly analysed for utility and performance in operation. The discussions and findings made, are presented in a conversant but clear manner. Innovations exposed by the artefact’s integral design, is emphasized for consideration in enhancing the body of knowledge.

To end with, **Chapter 6** summarizes the research contribution and suggests avenues of future work.

1.10 Chapter Summary

In this chapter, the salient problems faced in building business intelligence within a modern day data analytical environment was initially deliberated. A major concern for business intelligence building, was identified as dealing with capacity shortfalls. On-demand computing capacity generation, as provided by the distributed computing paradigm, could facilitate a potential solution to the capacity problem. Revealingly, entities are now increasingly looking to cloud technologies to provide such on-demand services.

The benefits that may be gained from the cloud and big data analytics, assuredly pose an attractive value proposition for organisations and the private individual alike. However, the move to cloud could increase the technical debt exposure and infrastructure obsolescence cycle for currently retained IT assets. An obligation is engendered to seek repurpose of IT assets, in a manner that remains aligned with the organisational need. Creating on demand grids from current and legacy IT assets, in the form of HPC or HTC computing platforms, could have potential fit to the immediate alignment issue.

Further discussions provided insights into the contrasting nature of HPC and HTC. The potential efficiencies for parallel computation at participant node level is a desirable aspect of HPC. Then also the cooperative data handling, scalability and fault tolerance found in HTC appeals. The surveyed literature overwhelmingly calls for a best of both worlds approach to HPC and HTC, though on reflection significant obstacles to the data analytical value proposition remains.

CHAPTER TWO
LITERATURE REVIEW

The research literature choice was informed by availability of the originally authored and peer reviewed documents in their entirety. The selected papers had to conform to the specific keyword domains and supplementary extents, as highlighted in the research endeavour. Popular scholarly search engines, IS conference proceedings and journals were employed as utility to acquire articles of interest. Only multi-cited or considered seminal author works were designated for inclusion within the review. Where required, additional technical specification or clarification was obtained from official organizational websites, white papers and tertiary educational textbooks. The rationale and need for secondary sources, stem from the fact that the research endeavour encompasses multiple diverse fields of academic study. The contained research primary and secondary domains within the review have been organized in Table 1. It should be noted that a definitive case of previous academic research having been undertaken to incorporate all comprised research domains, could not be made using prevailing resources.

Table 1 Research domains that inform literature choice

Primary Domains	**Sub Domains**
Design Science Research	Research and Systems theory
Grid computing & Heterogeneity	Abstraction Application environments Cache topologies and management CPU topologies Network socket environments Inter process communications Operating Systems Parallelism Security contexts Software optimization
High Performance Computing	Message Passing Interface
High Throughput Computing	Distributed File Systems Fail-over and fault tolerance Generic storage Remote Procedure Call
Technical Debt	E-waste Strategic information systems

2.1 Literature Study Selection Protocol

Preliminary results for article selection were obtained via search strings written in the English language, as inputted into online search engines and academic databases. Predominant database sources for articles have been presented for perusal in Table 2. The search strings comprised unary or combinatorial keywords as identified within in the research undertaking. The possibility of keyword alternate spelling or synonymic use was explored and actioned for pertinent relevance.

Table 2 Survey database sources

Database Source (EBSCOhost / Google Search / Google Scholar / Sage journals)	**% in use**
Institute of Electrical and Electronics Engineers (IEEE)	35%
Association for Computing Machinery (ACM)	18%
Springer	7%
Elsevier	7%
Other	33%

As part of initial selection, foundation articles had to expose best fit within the principles of peer review, contemporary authority and research domain bearing. Where possible, supportive literature reviews or assessments across discreet domain body of knowledge areas were sought to reveal additional articles of interest. The need for cross sectional supporting literature was deemed important, as justification for such could be established. Per illustration, it was recognized that title or keyword matching did not continually provide relevant papers for the selection process.

Nevertheless, only freely available and full text papers were eventually promoted for scrutiny. The explicit research applicability was determined by examining the article authors' abstract and closing statements. The preliminary papers obtained were then categorized by extent and consequently independently investigated. Within an academic domain extent, author viewpoints and recommendations from isolated investigations were mapped for consensus, enhancement or divergence.

The examinations conducted per isolated article reviewed, in certain instances produced supplementary atomic domain principles which required external enhancement. As shown in Figure 1, a sub process for selection was initiated in such circumstance, to augment core concepts or contextualize foundation narratives. The use of technical white papers, academic courseware and brand official websites were not excluded as instrument of such enhancement.

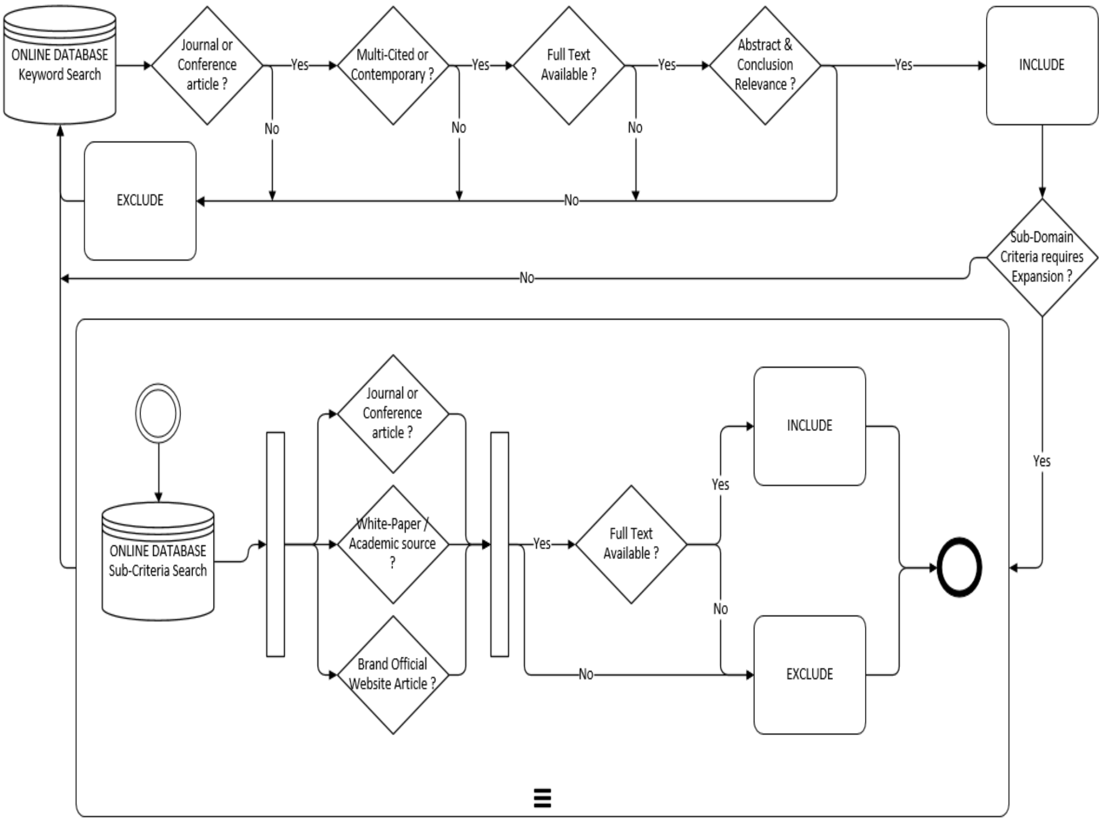

ONLINE DATABASE Keyword Search
Journal or Conference article ?
Yes
Multi-Cited or Contemporary ?
Yes
Full Text Available ?
Yes
Abstract & Conclusion Relevance ?
Yes
INCLUDE
No
No
No
EXCLUDE
No
Sub-Domain Criteria requires Expansion ?
No
Yes
Journal or Conference article ?
White-Paper / Academic source ?
Brand Official Website Article ?
ONLINE DATABASE Sub-Criteria Search
Yes
Full Text Available ?
Yes
INCLUDE
No
No
EXCLUDE

Figure 1 Literature selection logic process

The additional sources acquired by means of the sub process, were vetted to satisfy at least one of the following conditions:

- Multi-cited in topical academic literature
- Tertiary educational courseware (undergraduate or above level)
- Technical reference work or industry white-paper
- Internet based, official product website article

The overarching consideration for secondary sourced papers, regardless of the gauging qualification being the official or normative sanction of such material. The need to lend validity to claims made by imposing selection criteria on secondary sourced articles was perceived as warranted in reducing possible shifts with regard to burden of proof.

2.2 Technical Debt

2.2.1 A Historic Perspective on Technical Debt

The term *Technical Debt*, was first coined via an article in 1992 by Ward Cunningham titled: "The WyCash Portfolio Management System. OOPSLA' 92 Experience Report". Originally, the term technical debt is used by Cunningham as a metaphor. An allegory is generated to describe how stopgap actions targeting quality may impact holistic system lifecycles (Brown, et al., 2010; Klinger, et al., 2011; Martini, et al., 2014). The induced speed of development that marginalizes quality considerations is at the heart of the matter. Organizational pressures advocate quick win conditions that may weaken problem identification, investigation and analysis, design and development rigor. An organization plausibly perceives that cost and resource gains may be achieved by reducing the system's development effort and schedule. The decrease of effort and time associated with the endeavour, should directly constitute cost savings. However, the reality is, that similar to actual financial debt scenarios technical debt could arise. The organization would need to service the debt of maintaining and rectifying the low quality or faulty system as a result (Brown, et al., 2010; Klinger, et al., 2011; Martini, et al., 2014). The long term consequences for spontaneous operational or tactical actions, however originally defensible, may need to be considered. In this context, the debt metaphor has found popular contemporary traction, as it provides a collective understanding of the implications for rushed or ill-conceived decision making.

The main contributors of technical debt are often the non-technical stakeholders (Klinger, et al., 2011). Management decisions concerning financing and resourcing of system products are prepared without understanding the long-term implications of technical debt creation. The communication gap between technical and non-technical stakeholders then exacerbate the situation, by not articulating the implications and cost of incurring technical debt.

A mutual understanding of the term *debt* by both engineers and management, should ideally act as an enabler (Brown, et al., 2010; Klinger, et al., 2011;Tom, et al., 2013). The universal concept and implications of financial debt are typically well understood by all stakeholders. We take on debt naively or on purpose, but the debt needs to be paid sooner or later. The sooner we can service the debt, the less influence it has on our lives. The debt narrative may then be extended to relate to information systems, allowing debate amongst stakeholders.

The discussions generated from using a focused terminology such as technical debt, may reveal numerous topical aspects for consideration. As an example, consider a pre-development scenario. The technical debt features deliberated could very well provide inputs into pending venture decisions. In a post-development scenario, identified technical debt issues may be traced backwards to flawed conception. Demonstratively both view contexts provide information to stakeholders, which may enhance understanding and accumulate knowledge towards augmenting processes or organizational business intelligence. Several discourse benefits are highlighted for additional clarity (Brown, et al., 2010; Klinger, et al., 2011; Tom, et al., 2013):

- Discussion facilitates wider stakeholder involvement from all organisational spheres, revealing possible sources of technical debt that might otherwise lead to sub-optimal decision making
- Possible escalated total cost of ownership of a system can be identified early on
- Discussion generated, may expose currently unforeseen complexity in integration with existing systems that has bearing on the scope or resulting agility of the proposed system
- Improper business cases for systems may be pre-emptively discarded by quantifying financial implications of possible technical debt

Notably it should be understood that technical debt is fundamentally inevitable. A system would carry initial and residual quantities of technical debt, even after due diligence conceptualization, development or maintenance expenditure (Martini, et al., 2014). The repercussions then are that cumulatively over time and across the organization's system entities, the technical debt would eventually reach crisis proportions. Some technical debt may even have been entered into on purpose. The technical debt intentionally created for situations where the organization wanted to advance opportunities it may not have otherwise been able to afford (Klinger, et al., 2011). The crisis or tipping point descriptive can be important for continual organizational awareness in regards technical debt.

Principally the descriptive of technical debt could be used to galvanize support from stakeholders to ensure long-term value is generated from information technology resources on a persistent basis. At the outset the term technical debt, as described by Cunningham, was meant to refer to software development environs (Brown, et al., 2010; Klinger, et al., 2011; Martini, et al., 2014). The technical debt payable, due to non-holistic decision making since Cunningham's original publication, has however been found to be rather more multi-dimensional.

2.2.2 The Contemporary Technical Debt and the E-waste Link

The impetus by contemporary technical debt researchers is to extend the terminology understanding to a more comprehensive ecological form (Tom, et al., 2013; Betz, et al., 2015; Ernst, et al., 2015). Topically, an argument may be made that after all software executes on hardware and supporting infrastructure. To address the absence of a wide-ranging and academically sound scope for technical debt, a comprehensive taxonomy was proposed by Tom et al. (2013). Primarily the taxonomy is meant to be used as a technical debt exploratory framework. Elements listed by the Tom et al. (2013) taxonomy, consists of: dimensions, attributes, precedents and outcomes. The universal dimensions of technical debt are identified as: code, design and architecture, environment, knowledge distribution and testing (Tom, et al., 2013). Each dimension mentioned in the list, containing attributes of interest. The matrix when formed accordingly with the remaining elements, provides exploratory information between technical debt components and ultimately elaborate towards probable outcomes. As such, the environmental attributes when measured, may reveal additional debt burden which when factored provides for a better total accrued technical debt picture. In practice when considering hardware as an example, the obligation is then to also scrutinize the infrastructure and supporting applications for *architectural*, *environmental* or *sustainability debt* (Tom, et al., 2013; Betz, et al., 2015; Ernst, et al., 2015).

The functional demands influencing software over time will speed the eventual obsolescence of the hardware architecture that currently supports such (Fitzpatrick, et al., 2014; Remy & Huang, 2015). It is known that obsolescence is unavoidable in most cases as technology would continually advance. Remarkably, contributing factors cited for hardware obsolescence have both planned and unplanned associations. Planned obsolescence is informed by the effects of Moore's Law regarding the decrease in hardware observable lifecycles, due to technological advancement (Widmer, et al., 2005; Blevis, 2007; Remy & Huang, 2015). Unplanned obsolescence is importantly conversant towards flawed or inadequate design as a major contributor to the formation of hardware obsolescence (Blevis, 2007; Betz, et al., 2015; Remy & Huang, 2015).

Flawed or inadequate design and related decision-making inferences, have perceptible equivalents in the technical debt arena (Tom, et al., 2013; Betz, et al., 2015). Whether it be planned or unplanned obsolescence, the design of the information technology solution is incomplete without due consideration as to what would become of the system objects after loss of originally intended functionality (Blevis, 2007; Betz, et al., 2015; Remy & Huang, 2015). An entire field of academic research has evolved to study the obsolescence phenomenon more closely.

Academia use key words such as *e-waste,* short form for the word pairing *electronic waste*, to describe their efforts in the obsolescence arena. Importantly the e-waste generated from hardware retirement has long been viewed as an imminent global environmental threat (Widmer, et al., 2005; Blevis, 2007; Robinson, 2009; Fitzpatrick, et al., 2014; Betz, et al., 2015; Remy & Huang, 2015). The scholarly e-waste research focuses on causes, main contributors towards, physical outcomes and viable solutions to what is in fact literally an electronic garbage creation problem. Hardware and software are inextricably linked in the obsolescence cycle (Blevis, 2007; Robinson, 2009; Tom, et al., 2013; Betz, et al., 2015). An argument may be made that e-waste generation is a natural outflow of technical debt. Ill-conceived decisions around software have real physical outcomes in the form of hardware objects that may be unable to sustain functionality as intended.

Proposed solutions to technical debt, like the phenomenon itself, have multi-dimensional characteristics in literature. As an example, consider a specific and recurring theme in the code dimension of technical debt regarding the software development and maintenance setting. The specific technical debt solution suggests the continuous *refactoring* of code. The word refactoring, meaning the rewrite of system source code without influencing the intrinsic system's behaviour or functionality. The refactoring exercise itself however has complex proportions that impact the effectiveness of the technical debt reduction endeavour (Brown, et al., 2010; Martini, et al., 2014; Ernst, et al., 2015). Consider an additional example eluded to earlier, the technical debt solutions in topical literature which specifically address hardware concerns are initially found to be sparse or non-existent. On closer scrutiny however, technical debt articles promote the key dimensions of architecture and environment to encompass hardware objects (Tom, et al., 2013; Martini, et al., 2014; Ernst, et al., 2015).

The complexity generated when extruding all the dimensions of technical debt as provided for in literature, then builds conceptual appreciation for its non-trivial landscape. In mitigation of complexity, Martini et al. (2014) contends that all technical debt literature largely focuses around understanding the root causes, whilst attempting to reduce or delay the burden of payment.

The motivation around research aswell as contextual issues incurred within the subject matter of e-waste and technical debt have been found to be strikingly similar. Importantly, the proposed solutions to e-waste predominantly include (Blevis, 2007): *promoting renewal and reuse* together with the ability to *link invention with disposal*. The e-waste justification of promoting renewal and reuse, imitates the technical debt concept of refractoring. Linking invention with disposal as described in e-waste, has connotations of technical debt's ability to perform traceability to flawed design. The milieu of subject matter for both technical debt and e-waste, may be mutually inclusive. As to re-iterate this connection, Betz et al. (2015) draws special significance to human social responsibility around technical debt creation. The design decisions around information systems must consider sustainability or *sustainability debt* as a subset of the technical debt metaphor (Betz, et al., 2015). An ill-judged concession on information system design, could equate to compromising future generations in the form of e-waste. The technical debt phenomenon is consequently, amongst others not just a software development, management or process problem but also an ethical one.

The subscription to a more fixated definition of technical debt may be appropriate, particularly given its interconnected relationship with e-waste. Contemporary technical debt research articles reveal that a de facto definition for technical debt has as yet not been adopted. Due in part to the multi-dimensional nature of technical debt phenomenon, discussion in literature continues to redefine or augment the metaphor (Brown, et al., 2010; Klinger, et al., 2011; Tom, et al., 2013; Martini, et al., 2014; Betz, et al., 2015). To avoid ambiguity, the term technical debt would non-prescriptively henceforth describe:

> *The inability of information technology resources to be reconfigured or scaled due to short term non-holistic decision making, without regard for long term outcomes leading to potential loss.*

Initially, the derived definition should acknowledge the legacy perspective of what is undoubtedly known and understood to be technical debt. The primary aspects pertaining to people and their decisions with resultant tangible impacts needs to be reproduced. Moreover, the definition should apprise that the scope of technical debt contains amongst others, both the hardware infrastructure and software facets. Lastly the possibility of loss narrative, should provide motivation to potential stakeholders that the product of the debt metaphor may well be negative and thus of concern.

Academic literature surveyed does provide credence to the theory that e-waste would be produced on a near exponential scale within the next decade (Dwivedy & Mittal, 2010; Yu, et al., 2010; Petridis, et al., 2016). The proportional metric of e-waste production as an outflow of technical debt is currently indistinct. What is clear though, is that the speed at which hardware obsolescence is occurring may be caused by ever decreasing computer system lifecycles (Petridis, et al., 2016). Given current trends in ICT, it may be possible to hypothesize the extents of technical debt's physical outflows by investigating historic and predictive statistical data.

2.2.3 Current and Future Sources of Technical Debt

As shown in Figure 2 & 3, the initial thriving global computer sales market at the beginning of the century appears to now be in steady decline. The cause of the degeneration observed could justifiably be argued to be multivariate. The generation of plausible arguments would however require qualification by foundational principles.

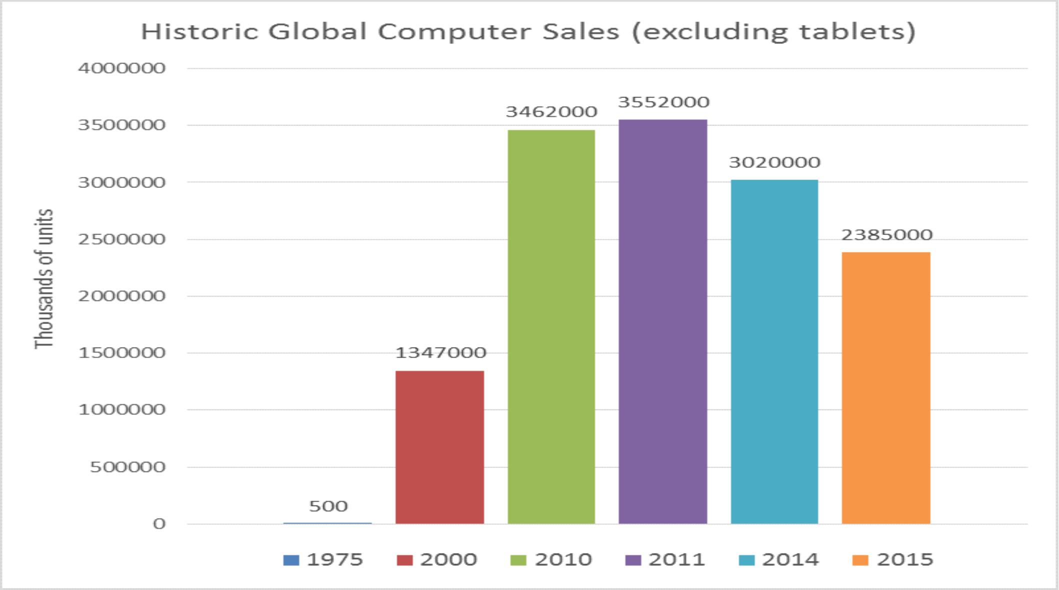


Figure 2 Historic computer sales data – inclusive of servers (adapted from Statistic Brain Research Institute, 2016)

To create a sound premise as to factors driving market environments, the use of Michael Porter's *Five Forces Model* is widely advocated (Grundy, 2006; Bidgoli, 2016). Well accepted by academia and practitioners alike, Porter's model provides for the analysis of market environments to facilitate understanding of underlying influences. The strategic abstraction and analysis of the market environment by use of the model, should provide insights that could enhance decision making.

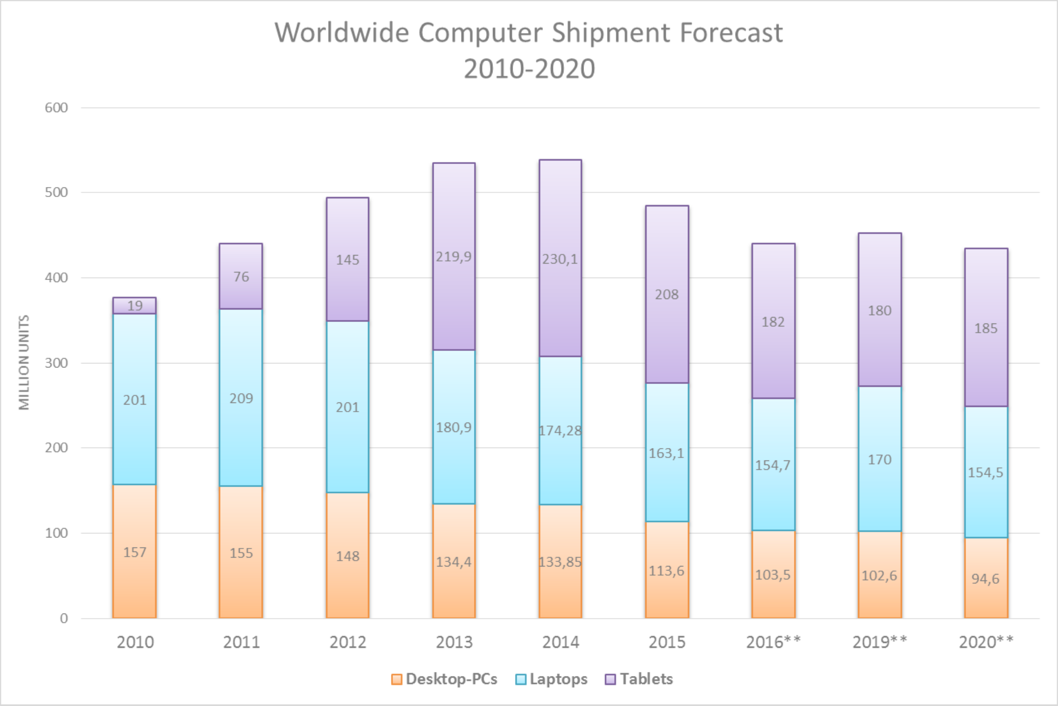


Figure 3 Computer shipment forecasts world wide (adapted from Statista Inc, 2016)

Applying Porter’s model in its original form to the observed global computer sales data, may create six immediate theoretical scenarios as shown in Table 3, which could explain the current downward trend.

Table 3 Evaluation of computer market data

1. Product saturation has been realized
2. A change in buyer behaviour has manifested
3. Affordability of the product has reduced buyer uptake
4. Reduced product or aggregate resulted in reduced sales stock
5. Substitute products have been introduced or technology shift has occurred
6. Rivalry amongst competitors have strategic product swing dynamics

Presumptively the first four derived theoretical scenarios may however be discarded as major contributors to the decline in global computer sales. In scenario 1, the known ubiquitous nature of computing as well as the growing demand for computing within an information age should preclude imminent saturation (Yu, et al., 2010; Bidgoli, 2016; Petridis, et al., 2016). An argument may be made for scenarios 2, 3 & 4 as possible bases, however it might not be able to adequately explain the measured trend of the decline observed. If such influences were apparent, more pronounced movement or demand dislocation over short periods of time would have been expected in the data plot. The proposition of possible substitute products and strategic focus swing amongst competitors, as described in scenario 5 & 6 are more likely candidates. The introduction and subsequent large-scale adoption of cloud computing and allied technologies could be just such substitute products, with accompanying organizational strategic focus implications (Botta, et al., 2016). The notion of cloud computing's influence on reduced global computing sales was perhaps provided for in recent commercial research. Per a Gartner Research Group prediction in 2016, the majority of IT spend by the year 2020 would be on cloud and related services. Cloud computing as platform and infrastructure alternatives to existing computing environments may well build a feasible case for the downward trend in current global computing sales. An additional factor could be that the uptake of virtualization technologies has impacted the quantity of computing systems sold. The advent of so called *smart technologies*, as exhibited in cellular phones, may have constituted viable computer system replacements and therefore have influenced the observed decline (Bidgoli, 2016). The important consideration in regards holistic technical debt and e-waste manifestations are that in practical terms, the potential for e-waste generation is being increased.

The computing platforms at risk for technical debt and e-waste creation, could further be informed by the global market for systems software. Application software necessitates execution via application programmable interface (API) interaction, as exposed by an operating system. The operating system in turn controls and interacts with the underlying hardware, which ultimately enables the application's functionality. As shown in Figure 4, the Microsoft™ Windows© platform at time of writing, holds much of the desktop system software market share at approximately 90%. Also noteworthy is that the data displays the natural decline and uptake cycle of replacement versions of the Microsoft™ Windows© operating system. The Microsoft™ Windows© (henceforth Windows) operating system, is completely proprietary and commercially licensed, on an effectively singular unit basis. Installations of an operating system are either physical, regard remote access or are virtual implementations (Mclean & Thomas, 2010; Microsoft Corporation, 2016; Microsoft Corporation, 2017).

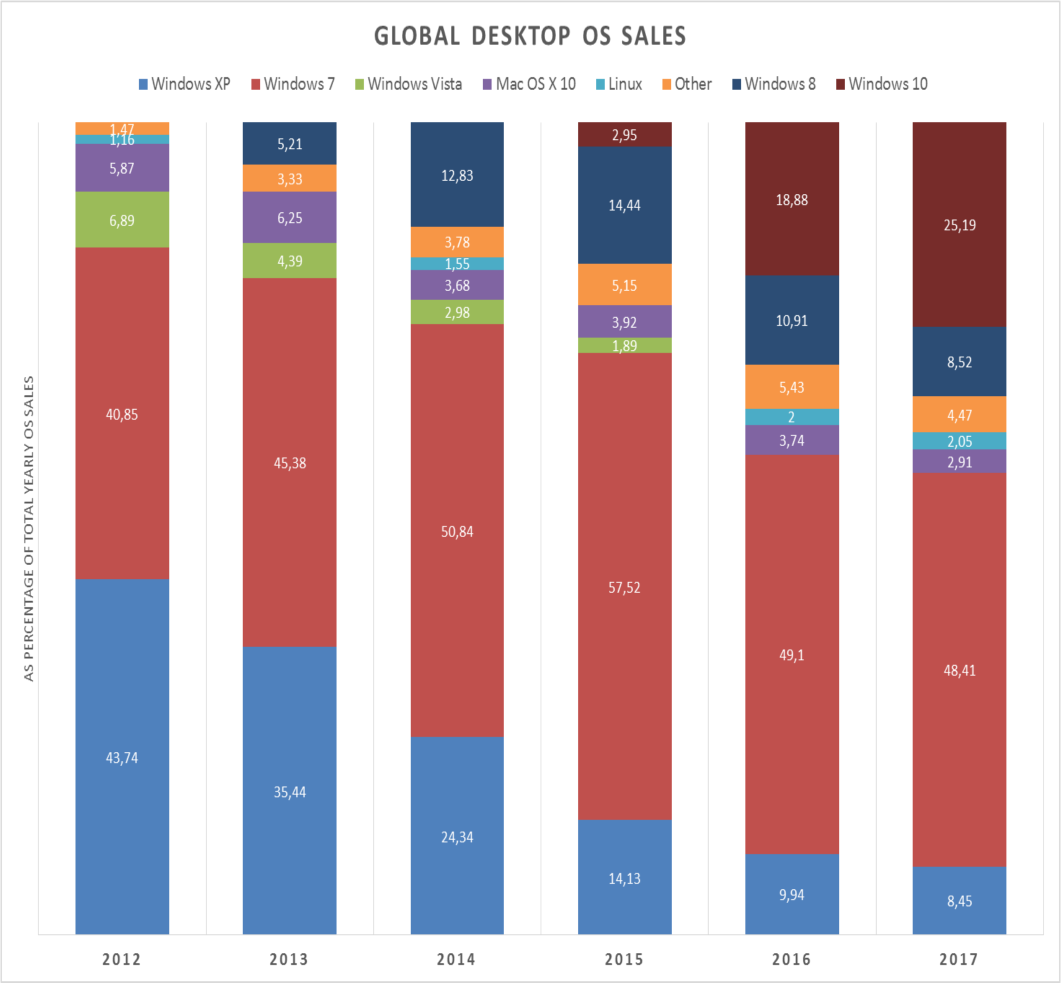


Figure 4 Global desktop operating system sales (adapted from Net Applications, 2017)

The licensing of Windows typically has additional restrictions imposed dependent on the version or edition under consideration. Significant restrictions of interest include (Mclean & Thomas, 2010; Microsoft Corporation, 2016; Microsoft Corporation, 2017): number of concurrent networkable nodes accessible; the number, type and size of processors as well as random access memory supported; restrictions around underlying hardware architecture, version and edition upgradeability. The elimination of choice by imposing such restrictions around operating systems, may possibly influence the speed of reduced functionality experienced by both software and hardware. As new minimum hardware requirements need to be met to support ever larger footprint operating systems, not only are previous operating system editions and versions being discarded, but conceivably also the existing hardware infrastructure. The interdependence of the infrastructure, hardware and software environment could play a significant role in an attempt to repurpose currently held organisational IT assets as HPC and HTC grid solutions.

2.3 The Operating System Construct

At the individual computing node level, the relationship between hardware and the operating system would dictate the functionality exposed. The compatibility of the operating system with its underlying hardware, forms the basis on which functionality through user application and configuration could ultimately be achieved.

Modern operating systems, as depicted in Figure 5, realize compatibility with underlying hardware via a scheme of detection, subsequent kernel compilation, together with exposing hardware abstraction layer and device driver instances for a given platform. The operating system construct is considered for all intents and purposes to be unique per installed node (Microsoft Corporation, 2003; IBM Corporation, 2008, Mclean & Thomas, 2010). The ability of the operating system to gain fine grained compatibility with the hardware environment, facilitates multi-manufacturer and therefore dissimilar hardware platform support. Ultimately the dissimilar hardware and operating system environments would speak to the *heterogeneous* nature of commodity computing platform realities.

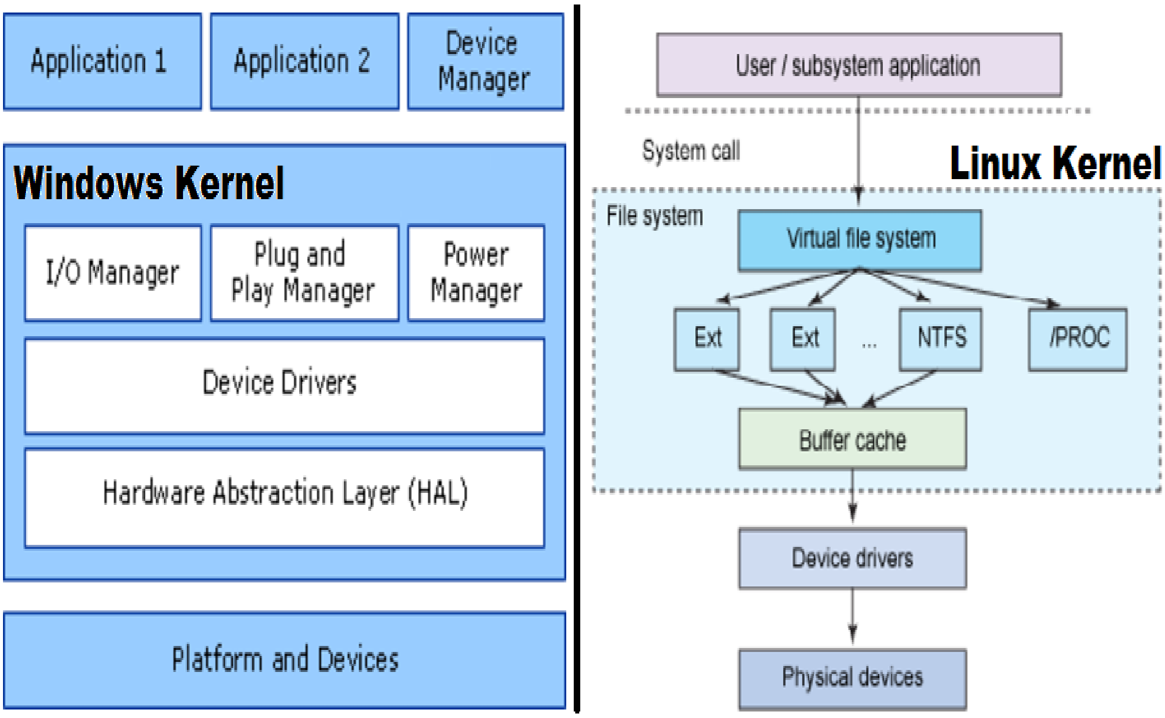


Figure 5 Modern modular operating system kernels
(adapted from Microsoft TechNet, 2003 & IBM developerWorks, 2008)

2.3.1 Application and Operating System Interactions

Although the ability of the operating system to conform to the hardware environment is assuredly desirable, the operating system through inherent or enforced architecture feature restrictions could expose quite wide-ranging functionality. As a consequence, to such wide-ranging execution environments, contemporary application development stereotypically focusses on *abstraction* via programming language and accompanying frameworks, in order to gain run time or cross platform compatibility. The term abstraction, meaning an emphasis on simplifying certain underlying system characteristics using some form of summarization, whilst suppressing details and properties of others (Shaw, 1980; Kiczales, et al., 1997). The use of application abstraction has however significant advantages and disadvantages for consideration. Abstraction would gain application code understand ability and maintainability, but inevitably sacrifice efficiency (Kiczales, et al., 1997; Berlin, et al., 2004; Akhter & Roberts, 2006; Newburn, et al., 2011; Rossbach, et al., 2013). Each layer of abstraction, to facilitate software to hardware interaction via the operating system, as shown in Figure 6, would add obligatory layered complexity and processing overhead. In opposition, a reduction in abstraction of the functional application has the potential to lessen processing overhead and deployment complexity whilst significantly decreasing wall clock time for discreet application runs. Normally the disadvantage for reducing application abstraction would be the necessity of recompilation, versioning and configuration management per platform supported (Shaw, 1980; Kiczales, et al., 1997; Rossbach, et al., 2013).

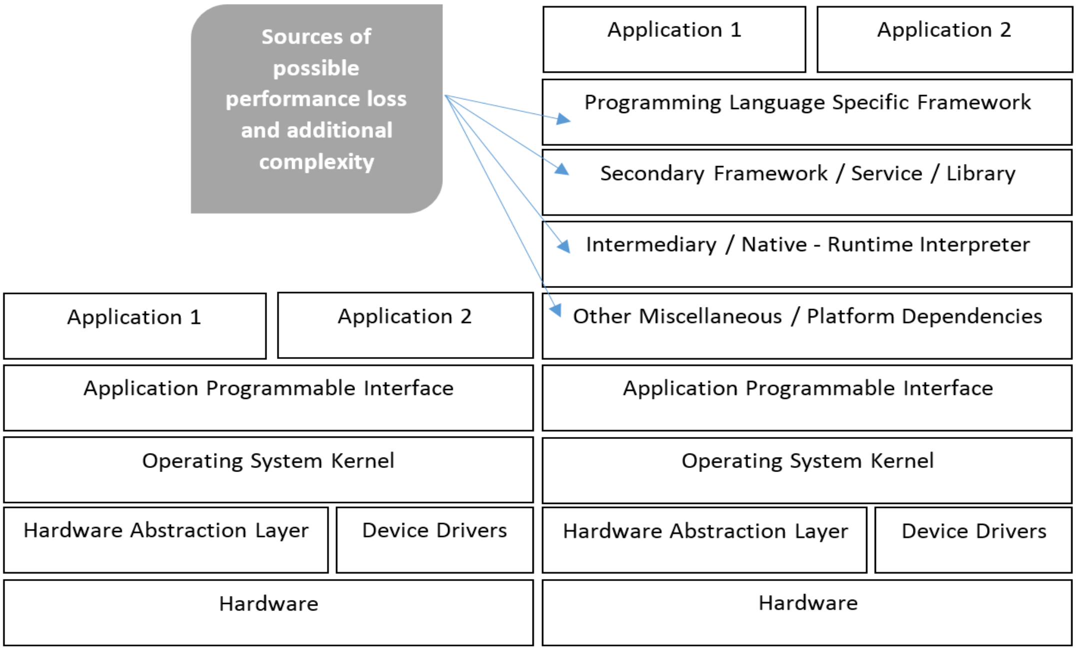

Sources of possible performance loss and additional complexity
Application 1
Application 2
Programming Language Specific Framework
Secondary Framework / Service / Library
Intermediary / Native - Runtime Interpreter
Application 1
Application 2
Other Miscellaneous / Platform Dependencies
Application Programmable Interface
Application Programmable Interface
Operating System Kernel
Operating System Kernel
Hardware Abstraction Layer
Device Drivers
Hardware Abstraction Layer
Device Drivers
Hardware
Hardware

Figure 6 Comparing technology stacks, highlighting abstraction

2.3.2 Abstraction and the Heterogeneous Environment

The accurate discovery of operating system, hardware and network topology environments would be key to any meaningful HPC or HTC implementation (Buyya, et al., 2000; Kennedy, et al., 2004; Youseff, et al., 2008; White, 2012; Dobre & Xhafa, 2014; Mantripragada, et al., 2015). Pertaining to previous deliberations around abstraction and application technology stacks however, a three-way trade off condition is created in that conflicting goals of the intrinsic endeavour could manifest (Berlin, et al., 2004; Kennedy, et al., 2004; Reyes-Ortiz, et al., 2015; Saxena, et al., 2016). The source of such trade-off conditions being conversant of the heterogeneous nature, as well as the programmable environment of computing platform realities:

- Goal 1- The implementing application should have best fit to the underlying hardware as to assure high efficiency
- Goal 2- The application should be portable across a variety of platforms as to increase possible functional exposure
- Goal 3- The application should be easy to create, use and maintain

In ensuring best fit with underlying hardware, the application would need to be abstracted as little as possible. However, to gain portability across heterogeneous platforms the use of abstraction would be unavoidable. Conversely for application development to be rapid, efficiency in execution would need to be compromised via additional layers of yet more abstraction. The current HPC and HTC application environments are built on numerous interactions of abstraction (White, 2012; Grossman, et al., 2013; Rossbach, et al., 2013; Gupta, 2015; Reyes-Ortiz, et al., 2015; Saxena, et al., 2016). An argument for the idea that influences on efficiency may occur due to abstraction within HPC and HTC implementations could well be justified. The HPC and HTC commodity heterogeneous platform realities, provide for difficulties when it concerns implementation choice and efficiency in performance (Xie, et al., 2012; Dobre & Xhafa, 2014; Reyes-Ortiz, et al., 2015). Assuredly the contemporary HPC and HTC implementations do overwhelmingly have the ability to execute on commodity hardware. Though on scrutiny of the fundamental design intents of popularly available HPC or HTC implementations, the target platforms display fundamental homogenous characteristics (White, 2012; Dobre & Xhafa, 2014; Microsoft Corporation, 2016; Open Science Grid, 2016; Intel Corporation, 2017; University of Wisconsin-Madison, 2017). The most notable supported platform design limitations were found to be: component brand and type of individual hardware components, operating system distribution, application frameworks, network fabric, supporting interface libraries and programming languages.

The literature surveyed further supports the notion of design homogeneity inherent in HPC and HTC. The terms *heterogeneous* and *commodity,* as it concerns the holistic computing architecture, may have found multivariate meaning in use. The commodity hardware compatibility of HTC per example, explicitly does not mean inexpensive (White, 2012). Interestingly then also, within the academic literature surveyed when providing metrics in support of findings, truly heterogeneous platforms are not actually considered.

The term heterogeneous as used by Grossman, et al. (2013) describes comparison of platform and not architectural dissimilarity within the potential functional application's execution environment. Although the believed and inferred intent by Grossman, et al. (2013) is to demonstrate heterogeneous architecture interactions within an experimental hybridized HPC and HTC solution, the heterogeneity paradigm is possibly not extended to its complete form. As a result, it seems a standalone experiment conducted on one homogenous platform is optionally compared with the same experiment on another architecturally different platform.

Due diligence inspection of findings by other HPC or HTC authors, reveals similar singular platform runs being used as data points in support of conclusions. Habitually authors within the published works surveyed omit the all-inclusive possibility of heterogeneity influences at the computational node level. The selected review papers within the last decade that do reference heterogeneity concerns for inclusion in HPC and HTC design constructs, have been catalogued for perusal within Table 4. It may be noted that no singular author captures all these heterogeneous concerns emphatically.

The oversight of excluding heterogeneous characteristics may have wide ranging impacts on efficiency and specifically performance. Assuredly heterogeneity must exist at all the technology levels per participant node, if solutions are claimed to be heterogeneous. Considering these and other corroborating actualities exposed by modern-day HPC and HTC implementations via inherent design, it may be plausible that uncontaminated generalizability might be an unrealistic pursuit.

Gaining computational high performance and/or throughput requires assumptions about abstraction tolerances, communication environment and heterogeneous platform architecture in all its varied forms. The assumptions required by the undertaking could viably preclude purest generalization.

Table 4 Possible node heterogeneity

Source of heterogeneous characteristics	Cited by Author(s)
Application support constructs & security context	Montero, et al., 2011 Moreton-Fernandez, et al., 2017 Rossbach, et al., 2013
CPU topology	Broquedis, et al., 2010 Buono, et al., 2014 Grossman, et al., 2013 Lee, et al., 2010 Moreton-Fernandez, et al., 2017 Nelson, et al., 2015 Newburn, et al., 2011 Rossbach, et al., 2013 Saxena, et al., 2016
Graphics Processor	Broquedis, et al., 2010 Grossman, et al., 2013 Lee, et al., 2010 Moreton-Fernandez, et al., 2017 Rossbach, et al., 2013 Saxena, et al., 2016
Memory storage attributes	Broquedis, et al., 2010 Dobre & Xhafa, 2014 Grossman, et al., 2013 Lee, et al., 2010 Nelson, et al., 2015 Rossbach, et al., 2013
Network protocol stack, link speed & bandwidth	Broquedis, et al., 2010 Chang, et al., 2014 Dobre & Xhafa, 2014 Nelson, et al., 2015 Shvachko, et al., 2010 Zhao, et al., 2014
OS edition, version & exposed API	Chen, et al., 2012 Dobre & Xhafa, 2014 Lee, et al., 2010 Nelson, et al., 2015
Storage type, size and availability	Chang, et al., 2014 Dobre & Xhafa, 2014 Nelson, et al., 2015 Shvachko, et al., 2010 Xie, et al., 2012 Zhao, et al., 2014

2.4 The Execution Environment

A cognizance of the ability to harness parallelism at all levels of a possible HPC or HTC implementation is important, as efficiency and performance correlations may be drawn (Berlin, et al., 2004; Lee, 2006; Broquedis, et al., 2010; Newburn, et al., 2011; Ali, et al., 2014; Buono, et al., 2014; Gupta, 2015; Mantripragada, et al., 2015; Nelson, et al., 2015; Reyes-Ortiz, et al., 2015; Saxena, et al., 2016; Moreton-Fernandez, et al., 2017). The onus therefore is to seek parallelism opportunities progressively whilst optimizing the use of data processing, communication and storage capabilities within the grid. The proper enumeration of the software application's execution environment should ensure effective and efficient use of available resources. In opposition the absence of environmental characteristics enumeration, could potentially create fatal dysfunction or suboptimal execution runs.

The inventorying of environmental resources could inform the idealized use, management and scheduling of resources as sought by the HPC or HTC venture. Moreover, the connectivity and interdependency of resources, together with dynamic avenues to possible integration may be gleaned. The foremost aim would be to uncover as many opportunities that could facilitate parallelism.

2.4.1 Parallelism, Concurrency and Threading

The similar terminology of *parallelism*, *parallel* or *concurrent* should not be misconstrued within the milieu of HPC and HTC. In single node concurrent computation, the operating system provides for the ability of software to simultaneously execute on a particular hardware processing resource (Akhter & Roberts, 2006). The simultaneous or parallel software workload execution achieved in concurrency might however mislead the unwary. Each discreet software workload or *software thread* at any given point in time, may only be executed exclusively per processing resource. In practice, multiple software threads are combined as an OS scheduled *process* or *instruction stream* (Alverson, et al., 1992; Akhter & Roberts, 2006; Majo & Gross, 2011). An instruction stream materializing the interleaved nature of the software threads to be executed. The operating system then designates a processing resource or *hardware thread*, as target for instruction stream execution. At any given point in time during execution, remaining software threads enter a wait state whilst a single thread executes. The environment of thread execution in concurrency therefore results in only one software thread making progress per interval. The interleaved software thread workloads merely imitate simultaneous execution and therefore do not constitute parallelism.

Eloquently in parallelism, multiple software threads concurrently make progress on different hardware resources simultaneously (Akhter & Roberts, 2006). The distinction then is that parallelism doesn't just leverage concurrency but also multiple physical processing resources which could potentially significantly enhance overall efficiency and performance.

The clarification of the terminology *thread* or *threading* might be required, when informed by the fact that reviewed literature surveyed could potentially obscure its meaning in use. Per illustration of possible opportunities for confusion, Lee (2006) uses the term to describe process sequences that share memory environments whereas Stratton, et al. (2008) uses the term logical threads to describe programmer functions or kernels that result in an execution thread blocks. Futhermore, Akhter & Roberts (2006) describes a thread, as a related sequence of instructions that excutes independantly of other instruction sequences. Conversely Buono, et al. (2014) and Dobre & Xhafa (2014) uses the term threading to describe communication thread interplay and thread provisioning in hardware. As a central principle, unity in denotation of threading is however found to be concrete. The dominant implications of threading, should be understood to be the smallest encapsulated construct that can be individually managed. It might nevertheless be prudent in order to avoid amiguity, that the term be explicitly contextualized as either being hardware, kernel, communications or software threading.

The efficiencies at node level may further be improved by considering how instructions and data are processed. The individual separate data sources or *data streams* which instruction streams operate on, could be leveraged for additional parallelism (Akhter & Roberts, 2006; Lee, et al., 2010; Jeong & Lee, 2012; Sabharwal, et al., 2013; Fog, 2017). Three distinctive machine environments have evolved to describe the relationship between data and instructions as found in commodity hardware, namely:

1. A serial machine construct that innately does not support parallelism, executing a single instruction stream that operates on a single data stream (SISD).
2. A single instruction and multiple data (SIMD) vector machine executing a single instruction stream, operating on a multiple separate data streams simultaneously. SIMD machines support parallelism at data level by operating on multiple data elements per executing instruction.
3. A multiple instruction stream and multiple data stream (MIMD) machine. A MIMD machine platform has multiple processors, each supporting separate instruction streams working on multiple separate data streams. MIMD machines therefore support parallelism at instruction and data level.

The major instruction sets available in commodity hardware, which provides for data parallelism via SIMD and MIMD, have been categorized in Table 5.

Table 5 Instruction sets supporting data parallel operations

Instruction Set	**Descriptive**
3DNow!	Packed single-precision, integer & floating-point vectorization
3DNow! Plus	Enhances 3DNow! by adding digital processing features
AVX	Advanced vector extensions, enhances MMX & SSE inclusive of additional support for: Advanced Encryption Standard (AES), FMA, packed carry-less multiplication
FMA	Fused Multiply and Add, A = A * B + C
MMX	Packed single-precision, integer vectorization
SSE	Streaming SIMD Extensions, supporting up to packed double-precision, integer & floating-point vectorization

2.4.2 The Central Processing Unit

The speed, type, number of processors, related cache hierarchies and facilitating instruction set supported on a CPU, could potentially significantly impact application functionality and performance (Advanced Micro Devices, 2005; Akhter & Roberts, 2006; Broquedis, et al., 2010; Jeong & Lee, 2012; Kusswurm, 2014; Intel Corporation, 2016; Kuo, 2017). The discovery of the CPU capabilities at computational node level would be important in determining the capabilities an application could exploit for parallelism.

The enumeration of CPU attributes and features however, should be considered non-trivial given real-world commodity heterogeneous platform realities.

At grid node level, the computing platform could either have a single CPU (known as a *package*) or support multiple CPUs. Each package may then support a singular processor (known as a *core*), or multiple physical processors (Advanced Micro Devices, 2005; Akhter & Roberts, 2006; Intel Corporation, 2016; Kuo, 2017). Additionally, each core could support simultaneous hardware multi-threading technology, as shown in Figure 7, revealing supplementary *logical processors.* The term logical processor meaning two addressable processors sharing the same physical processor resource environment (Nakajima & Pallipadi, 2002; Akhter & Roberts, 2006; Gepner & Kowalik, 2006). Fundamentally each logical processor could be sharing nearly half of the resource capabilities of the physical core (Nakajima & Pallipadi, 2002; Akhter & Roberts, 2006). Arguably significant performance disparity would be intrinsic when comparing logical processors with their physical core counterparts. The latent performance loss due to simultaneous hardware multi-threading is however in practice largely avoided. An operating system would only need to distinguish between logical and physical processors in designating execution workloads to unlock potential performance gains. Modern operating systems use a mechanism known as a *scheduler* to determine the optimum placement of software threads onto logical or physical processors (Nakajima & Pallipadi, 2002; Akhter & Roberts, 2006). The OS scheduler monitors and load-balances processor resource bandwidth to achieve higher efficiencies.

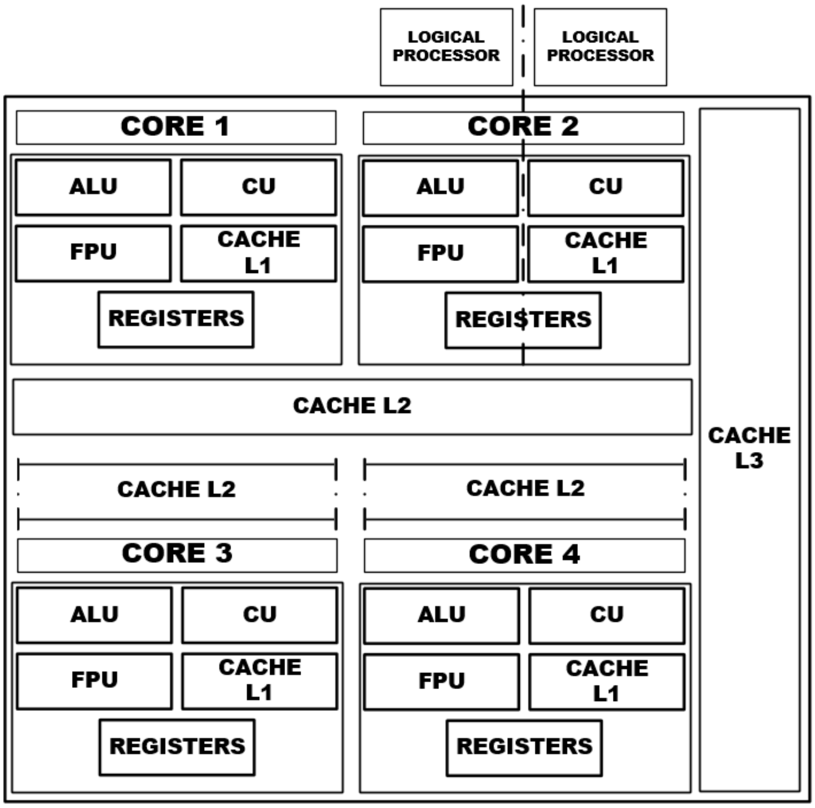


Figure 7 Conceiving a CPU package topology

In theory an application may even interact with the OS scheduler to suggest resource requirements or elect workload targets across physical or logical processors. Principally an application could bind its specific workloads to a discreet set of processors using an OS scheduler apparatus known as *affinity* and manipulate workload precedence, known as setting *thread-priority* (Nakajima & Pallipadi, 2002; Akhter & Roberts, 2006; Kazempour, et al., 2008; Broquedis, et al., 2010; García-Dorado, et al., 2013). The exploitation of fine grained threading should notably enhance holistic application performance. However, when linking threading to the topic of abstracted or *managed* applications, the conclusions are that such applications seldom have the ability to leverage threading (Akhter & Roberts, 2006). A managed application would have to rely exclusively on the operating system or additional run-time libraries to manage threading and in so doing, sacrifice possible efficiency gains.

2.4.3 Cache Memory Architectures

The cognizance of cache hierarchies and interactions within the CPU, play a vital role in possible application performance gains (Smith & Goodman, 1985; González, et al., 1995; Suh, et al., 2004; Akhter & Roberts, 2006; Majo & Gross, 2011; Kusswurm, 2014; Intel Corporation, 2016; Fog, 2017). The term *cache* describing a small fast memory type that endeavours to accelerate a slower but larger memory type's operations. The throughput from interactions with cache should therefore always be much higher than could be achieved via the alternate memory store (Smith & Goodman, 1985; Drepper, 2007). The internal speed of a CPU is typically faster than the variable external speeds of other hardware components within the computing platform. As rudimentary steps in the processing cycle the CPU requires *prefetching* instructions and data from random access memory (RAM), then also storing or *flushing* resultant data after processing. The speed discrepancy or *latency* between RAM and the CPU would inevitably cause processing stalls and resultant wasted processor cycles if not somehow mitigated.

The mechanism used by cache is to anticipate CPU memory access by prefetching data, thus not just improving data *locality* but also reducing potential latency (Smith & Goodman, 1985; González, et al., 1995; Suh, et al., 2004; Kusswurm, 2014; Fog, 2017). The term locality expressing the proximity of data and instructions for immediate CPU access. A CPU may even expose cache-ability control instructions, which allow an application to formulate a cache strategy to influence data access locality. The three cache locality data patterns of interest which may guide a conceivable application caching strategy, are listed as: *temporal*, *spatial* and *non-temporal* locality.

In abridged terms a cache prediction strategy would involve CPU near future data access, in terms of (Drepper, 2007; Kusswurm, 2014; Intel Corporation, 2016):

- Temporal locality, the same data elements previously accessed will require access again
- Spatial locality, the adjacent data elements to those previously accessed will require access
- Non-Temporal locality, the data accessed previously will not be accessed

A successful data access reference to a specific cache store which results in instructions or data being interpreted by the CPU, is known as a *cache hit*. Alternatively, if no reference is found, a *cache miss* occurs (Smith & Goodman, 1985; Kusswurm, 2014). As cache stores are accessed in sequence of availability from highest level and proximity to lowest level before obligated to access physical RAM, data locality and resultant latency caused by cache misses become important. An application could undertake to reduce cache misses and in so doing, increase processor efficiency. Any cache system memory store is a contiguous, power of 2 set of memory *blocks.* The prefetching and eviction of cache blocks are determined by a mechanism known as the *replacement policy* (Smith & Goodman, 1985; González, et al., 1995; Suh, et al., 2004; Kusswurm, 2014). The number of CPU accessible cache stores, their type and size could be crucial information an application could therefore leverage to gain processing enhancements. An application may well attempt to manage instruction and data blocks via means of internal build or the interface apparatuses of the replacement policy to avoid cache misses (Smith & Goodman, 1985; González, et al., 1995; Suh, et al., 2004; Intel Corporation, 2016; Fog, 2017). The CPU and memory interaction characteristics which may impact or inform performance considerations are further supplemented by *cache associativity*. The term associativity relating to how memory blocks are mapped from RAM to cache, as shown in Figure 8. Tellingly the associativity of cache would have direct bearing on data locality patterns during processing. Originally the mapping for a particular cache store was said to be either *direct mapped*, *fully* or *set associative* (Smith & Goodman, 1985; Drepper, 2007). Primarily in direct mapped associativity, each discreet entry or *cache line* may only refer to a specific RAM memory location. Whilst with a fully associative caching scheme, each cache line may refer to any RAM memory location independently. Finally, in a set associate scheme, cache lines are grouped into a number of sets, with each set capable of caching a specific memory area in set number of ways. Each caching scheme has its advantages and disadvantages in tangible implementation. Over time however, set-associate caching and variations thereof predominated, for having better real-world application fit.

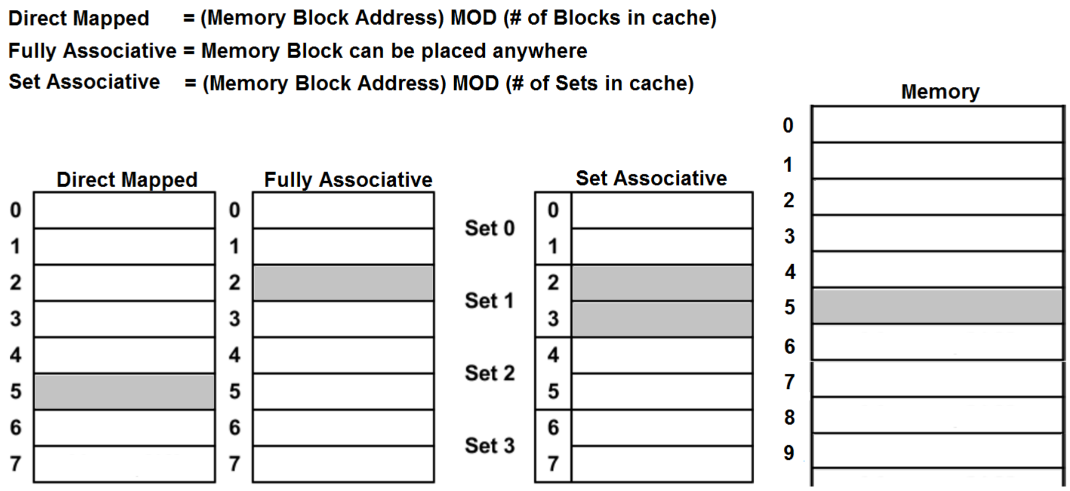

Direct Mapped = (Memory Block Address) MOD (# of Blocks in cache)
Fully Associative = Memory Block can be placed anywhere
Set Associative = (Memory Block Address) MOD (# of Sets in cache)
Memory
Direct Mapped
Fully Associative
Set Associative
Set 0
Set 1
Set 2
Set 3
0
1
2
3
4
5
6
7
8
9

Figure 8 Cache associativity

The management of cache and program blocks by a cache aware application are known to positively influence time critical processing (Drepper, 2007; Kazempour, et al., 2008; Advanced Micro Devices, 2014; Kusswurm, 2014; Intel Corporation, 2016; Fog, 2017). At nearest layer to the CPU, primary or *level one* (L1) cache is smaller and faster than supplementary caches. Per illustration, the speed of interaction between a physical processor and L1 cache approaches zero wait state, having little to no resultant latency (Drepper, 2007; Fog, 2017). Conversely, the *level two* (L2) and supplementary cache stores would be larger than L1 but would incur higher latency in access. In modern CPU topologies L1 cache is sectioned or *split* equally into *instruction* and *data caches* (Smith & Goodman, 1985; Advanced Micro Devices, 2005; Intel Corporation, 2016; Fog, 2017). Relating to previous discussions, recollect that such spilt cache at L1 would impact simultaneous multi-threading enabled platforms, as subdivision of the cache would ensue for each addressable logical processor (Nakajima & Pallipadi, 2002; Drepper, 2007; Intel Corporation, 2016; Kuo, 2017). The platform detected L1 cache resources, could therefore potentially be halved for simultaneous multi-threading implementations. The L2 and supplementary caches are stereotypically termed *unified caches*, that do not distinguish between data and instructions blocks. The L2 and lower caches may also display partitioning characteristics in that sharing of a cache store may manifest across physical or logical cores and packages (Advanced Micro Devices, 2005; Kazempour, et al., 2008; Intel Corporation, 2016; Fog, 2017). The sharing of cache stores amongst processing resources and resultant potential performance impacts can only be informed by CPU model specifications. A cache strategy targeted for homogenous CPU environments would therefore be a singular effort, whereas a hetrogenous environment requires a more multipronged approach.

2.4.4 Cache Optimization and Management Strategies

The most generic cache optimization technique necessitates that code instructions and the data it operates on, at least fit into L2 cache (Suh, et al., 2004; Akhter & Roberts, 2006). If multi-core processors are available, performance may be improved by shifting software threads to exploit caching architecture via processor affinity (Drepper, 2007; Kazempour, et al., 2008; Majo & Gross, 2011; Majo & Gross, 2013). Other equally universal suggestions that could potentially improve cache utilization are briefly listed as (Advanced Micro Devices, 2014; Intel Corporation, 2016; Fog, 2017):

- Utilizing the same memory operand sizes consistently
- Use PREFETCH instructions to hide bus bandwidth latencies in sequential, irregular or very large memory access
- Memory alignment of code, stack and data segments
- Do not store data in code segments or use self-modifying code
- Interleave SIMD type instructions in a Load-Store pattern

The application caching techniques proposed for performance enhancement, would however have nominal or even adverse effect when not considered within the context of *thread contention, cache thrashing* and *pollution*. Importantly the memory access patterns for individual software threads running on the same or separate processors, could implicitly create shared memory reference conflicts (Herlihy & Moss, 1993; Suh, et al., 2004; Eklöv & Hagersten, 2010; Herlihy, 2010; Sandberg, et al., 2010; Majo & Gross, 2011; Seshadri, et al., 2012; Majo & Gross, 2013). Thread contention occurs when separate software threads read and modify the same memory area. In circumstances of thread contention, cache thrashing could occur as large blocks of high use data would start evicting each other within the confines of cache size. Cache pollution occurs in situations where useful data is evicted and overwritten by non-useful data, causing subsequent references to such useful data to be reloaded.

Presumptively the interaction of a software thread's access would require data to be written or read from RAM via the cache hierarchy. The ensuing modification and access of a shared memory areas by multiple software threads could cause incoherent memory states during the prefetching and eviction cycles. Although CPU instructions potentially could allow software threads to bypass certain caches, pollution and resultant contention is all but inevitable if not somehow managed.

The efforts by a section of reviewed authors within the theme of cache management, suggests modification in algorithmic circuitry logic for multi-core systems to reduce cache pollution and thrashing (Herlihy & Moss, 1993; González, et al., 1995; Suh, et al., 2004; Seshadri, et al., 2012). The probable worth of topical algorithmic solutions are varied but seemingly undeniable in alleviateing caching issues. Steroetypically though within the surveyed papers, simulated evidence is provided in support of recommendations. The use of simulation debatably constitutes a potential theoretical solution, which by definition may have unclear pratical application characteristics. Undoubtably it would seem an unrealistic endeavour to modify hardware circuitry across existing hetrogenous platforms as a means to implement such algorithmic caching logic. As a consequence of the research context, adoption of such hardware algorithmic cache circuitry logic solutions may be excluded from consideration.

The management of the adverse effects of cache thrashing and pollution could potentially be viewed as a software design problem (Eklöv & Hagersten, 2010; Sandberg, et al., 2010; Majo & Gross, 2013). In reference to previous deliberations, a programmer might merely be required to sequence instructions and data in grouped blocks that favourably fit in higher level caches. Additionally, the programmer could produce code sequences with subsequent data access, which do not overtly evict useful blocks prematurely (Eklöv & Hagersten, 2010; Sandberg, et al., 2010). Nevertheless, the control exerted by the programmer would only extend to the thread contexts of the self-authored application. As the execution environment undoubtedly contains resource competing application threads, a cache strategy tediously implemented may have diminutive global impact. The endeavour to implement a caching strategy would have complexity dimensions beyond a singular application build, as mixed independent application workload realities may negate planned performance goals.

A work-around technique that targets mixed workloads of independent applications, relies on the pre-emptive profiling or classification of applications to recognize performance impacts and dependencies (Eklöv & Hagersten, 2010; Sandberg, et al., 2010). Within such a mixed workload environment of independent applications, a probabilistic model of cache misses may be generated for discreet short application runs. By monitoring the reuse distance of sparse and randomly selected memory references, reasonably accurate cache performance predictions can be made for differing cache topologies (Eklöv & Hagersten, 2010).

The classification of an application within a mixed workload also provides an opportunity to inject cache bypass instructions directly into the binary executable to reduce performance degradation of future execution runs (Sandberg, et al., 2010). Initially however this classification technique would be reliant on profiling the application within a specific and contextualized mixed workload environment. The effort expended towards the actual application classification also inducing varying supplementary overhead. Although such overhead is reported to be marginal and bespoke for a particular platform, best fit for truly heterogeneous platforms becomes indistinct. The predetermination of application context within a specific mixed workload scenario and the compulsory additional overhead imposed, may be considered undesirable or impractical. The OS scheduler itself could per example negate potential performance gains of the technique, due to its own internal logic attempts at reducing thread contention (Majo & Gross, 2011). The notion of predetermining the mixed independent workload environment, the subsequent modification and compilation of targeted application executables for a platform, is questionably a homogenous solution. As the truly heterogeneous memory environment contains indeterminate execution actualities, a sub-optimal cache management strategy could result from the profiling and classification technique.

The significance of classifying or profiling an application, should however not be discarded out of hand, since it could provide invaluable understanding of application's run-time behaviour. The understanding generated from profiling and diagnostics, would notably create opportunity in aiding application performance enhancements. As can be evidenced by the pervasive nature and uptake of commercial, as well as open-source profiling products, the software engineering community has had long standing appreciation of such in use. Some contemporary examples of profiling tools are: Intel® Vtune™ Amplifier & Advisor, Rouguewave® ThreadSpotter™, AMD® / CodeXL, DynaTrace™, Microsoft® Visual Studio™ Toolbox.

The emphasis for predictive profiling and classification falls into the realm of functional software development, when considered in the context of heterogeneous memory environments. The pre-optimization of internal application logic and cache usage via means of profiling is therefore not an all-inclusive exercise. The monolithic application construct developed from classification and profiling, importantly cannot by itself provide sureties of performance within arbitrary workload, shared and non-uniform memory (NUMA) platforms (Majo & Gross, 2011; 2013; 2015). A more dynamic and real-time control system would be needed beyond internal functional build, to regulate and normalize caching performance across heterogeneous systems.

A promising solution aimed at heterogeneous single threaded workload environments, that share last level caches (LLC), requires the combination of process scheduling and memory management. The basic premise of a coupled solution aims to curtail cache contention whilst maximizing data locality (Majo & Gross, 2011; 2013; 2015). Confronted by the feasibly conflicting objectives of cache contention and data locality, the former is said to take precedence in ensuring higher overall performance. The authors' Majo & Gross, (2011) proposed instituting a three-phase algorithmic process, baptized N-MASS as depicted in Table 6, to realize higher overall performance coupling.

Table 6 The N-MASS algorithm

N-MASS algorithm - adapted from Majo & Gross, (2011)	**Output**
Phase 1: Sort application processes by NUMA penalty NUMA penalty = $CPI_{remote} / CPI_{local}$	Per processor sorted lists
Phase 2: Calculate maximum local mapping	Maximize data locality
Phase 3: Refine maximum local mapping	Reduce cache contention

The N-MASS algorithm is mainly reliant on calculating the parameters of cache misses for an application process or *cache pressure*, together with a possible *NUMA penalty* (Majo & Gross, 2011). The term cache pressure denotes the *MPKI* or cache misses per 1000 instructions, as observed in LLC during execution of a single threaded application process. The NUMA penalty parameter, being the estimated ratio of execution cycles per instruction (CPI) of an application process when measured locally versus remotely (Majo & Gross, 2011). The term local in this context, denotes the first addressable logical processor on a CPU core which could spawn an application process and its data. The local application process is then said to be *homed* on the specific CPU core. Conversely remote execution of an application process, disregarding data placement, takes place on the next addressable logical processor on a different core. The salient theme is that single threaded application processes could potentially be sharing CPU cores and LLC. When inferring a heterogeneous CPU topology, the discovery of parameters in regards cache pressure and NUMA penalty could direct best effort towards placement of application processes. In the first phase of N-MASS, sorted lists of NUMA penalties in descending order are obtained for application processes. During the second phase, the sorted lists are combined and process mapping onto physical CPU cores with underlying logical processors is conceived. The essential assurance is that application processes that exhibit higher NUMA penalty, are mapped with higher priority than low NUMA penalty processes when homed on the same core.

In the last phase of N-MASS, the cache contention is reduced by comparing the process cache pressure against a predetermined threshold. If the cache pressure is higher than the threshold, a process may be shifted to achieve better contention balance whilst still favouring data locality.

The N-MASS algorithm, as originally conceived by Majo & Gross, (2011) does however expose several major limitations. Firstly, an assumption must be made that the functional application processes will not exceed the available number of logical processors (Majo & Gross, 2011). The implications being that N-MASS can account for the spatial multiplexing of a set number of application processes to maximize memory use. However, N-MASS cannot assure maximization without information interchange with- and exercising control over, the default OS scheduler. Secondly, the algorithm is reliant on telemetry access to the CPU's *PMU* or performance monitoring unit (Majo & Gross, 2011). The difficulty is that certain operating systems deem PMU directives to be kernel privileged or ring zero instructions, which may therefore require additional vendor signed kernel mode drivers. Thirdly, N-MASS does not address the impacts of migrating process data across cores (Majo & Gross, 2011). Although the algorithm considers the overhead and performance penalty of shifting the application process based on the threshold value, no such deliberations for data are catered for. Lastly, the applicability of N-MASS for multi-threaded processes becomes unclear as cache sharing could potentially improve performance in such instances (Majo & Gross, 2011). Hence the potential shared address space of multi-threaded applications, continued to make the mapping and scheduling of processes across specifically NUMA environments difficult. Only the later work by Majo & Gross, (2013; 2015) determined that several supplementary and major factors were at play for NUMA environments, as it concerned multi-threaded workload performance. As also previously discussed in this research paper, the specific data access patterns of software threads for a functional application may be determined by profiling and characterization. Subsequently the data access pattern, together with an understanding of modern CPU hardware prefetcher mechanisms, could then be used to make source code changes, facilitating the reduction of next-run contention (Majo & Gross, 2013). The important contribution by the authors, highlighting the role of the prefetcher and its causal effects on NUMA cache placement for multi-threaded workloads.

Lastly, proper configuration of process affinity scheduling with identity mapping, can associate software threads beneficially to physical cores and underlying logical processors (Majo & Gross, 2013; 2015). Be the nature of software threads symbiotic or autonomous, data access shared or independent, the process placement is key.

2.4.5 Random Access Memory and Secondary Storage Interactions

The computing node's physical RAM availability is yet another crucial element of interest, which could potentially be controlled to enhance overall efficiency. An envisioned grid node's RAM resources would constantly be in a state of contention. The operating system, background libraries and services, component input/output (I/O) buffers as well as running applications would all vie for immediate CPU accessible storage. When exploring the survey literature around memory resource concerns however, appreciative understanding beyond mere RAM availability is gained.

The traditional instrument used by an OS to curb the potential disparity in storage requirements and thus alleviate contention to physical RAM, is via the use of secondary storage augmentation (Ousterhout, 1982; Mclean & Thomas, 2010; Microsoft Corporation, 2013). The logical extension of primary storage by means of this type of OS functionally would fundamentally create a virtual storage area larger than physical RAM, but still accessible as a singular unit.

The modern OS manages primary storage contention by exchanging data from RAM to secondary storage and visa-versa. This mechanism employed by the OS to increase CPU accessible storage, is frequently termed *swapping* or *virtual paging* (Li & Hudak, 1989; Mclean & Thomas, 2010; Microsoft Corporation, 2013). For practical efficiency purposes, obvious concerns that stem from the inequality in access speed as well as bandwidth between primary and secondary storage would need scrutiny. A bottleneck scenario that informs potential reduced system efficiency by means of virtual paging may become apparent. The term *thrashing* in this context, describes a running application stall or *page fault* due to the CPU requesting access to memory which was previously swapped to secondary storage (Ousterhout, 1982; Li & Hudak, 1989; Mclean & Thomas, 2010; Microsoft Corporation, 2013). Page faults and subsequent disk thrashing, occurs when the amount of actively running processes' memory requirements exceed the size of physical RAM, forcing potentially extensive secondary storage access.

During disk thrashing, ensuing secondary storage access generates high processing latencies, which may result in a pseudo unresponsive system state. A system could become especially prone to disk thrashing in parallel processing environments (Ousterhout, 1982; Li & Hudak, 1989; Sabharwal, et al., 2013; Sharmilarani, et al., 2017). It should be understood then that disk thrashing is a symptomatic outcome of real-time machine load.

One dialogue for competing software threads that cause disk thrashing and I/O stalls, suggests utilizing larger buffer block sizes, consolidating read/write operations into a single thread, defragmenting the file system as well as using native asynchronous command queing (Sabharwal, et al., 2013). The overarching assertions being that not only would system performance increase, but also energy usage decrease by use of these techniques. Using larger buffer block sizes, above eight kilobytes for large sequential file transfer is said to require less processing resources and energy. The consolidation of read and write operatations into a single thread reduces contention, thereby increasing run-time performance. Defragmenting the underlying file system reduces the effect of secondary storage operational latencies. Asynchronous command queing reduces I/O blocking events thereby isolating read and write actions.

Although the mentioned disk thrashing mitigation techniques, by means of presented evidence are assuredly usefull, it may however on inspection raise concerns surrounding the specific mechanism of enactment. As presented by Sabharwal et al, (2013) these mitigation techniques require access to detailed hard disk drive metrics as input into the solution decision. The gathering of detailed hard disk metrics must arguably incur additional management overhead which is not emphatically addressed. Likewise the supposition that secondary storage devices would be physical platter based hard disk drives, becomes problematic within the research context of heterogeneous grid environments. The prescriptive use of detail level disk metrics such as rotation latency and revolutions per minute, could debateably skew the potential solution decision logic for non platter based systems. Not discounting the validity in implimentation of these generic techniques, nor the value of hard disk metrics, the consistant accuracy and the lowering in performance penalty due to overhead should be idealized.

The use of page faults as metric indicator is much more useful in determining workload conditions than reliance on pervasive or free storage metrics alone (Sharmilarani, et al., 2017). Per illustration the amount of free memory and disk space, does assist in formulating a decision path towards a feasible disk thrashing solution. However, whether used in isolation or combined, the free memory and disk metrics have no additional value in determining pressure on the system memory resources. The inclusion of system page fault counts, due to its weighting characteristics are better suited in formulating decision avenues in regards system workloads. Advantageously operating systems normally do support easily accessible, low performance penalty telemetry, whereby page fault count of selected or overall system processes may be acquired.

The page fault count when combined with the other specific metrics of CPU utilization and coarse-grained disk I/O channel throughput, would at minimum provide enough information to paint a reasonably accurate picture of system state (Sharmilarani, et al., 2017). Based on the prevailing system state, an informed solution may either terminate overloading processes, reschedule or redeploy workloads across less pressurized resources. Such mitigating actions would only be feasible by monitoring and reporting the three basic resources mentioned, on a per grid node basis. Subsequently the directive to redeploy or reschedule a discreet process under review, would need to be made by some form of scheduling mechanism.

2.5 Networking the Grid

The grid formation and clustering of computing resources in either HPC or HTC, is facilitated via *socket* based derivative network communications (Speight, et al., 2000; Romanow & Bailey, 2003; Broquedis, et al., 2010; Shvachko, et al., 2010; White, 2012; Chang, et al., 2014). Figure 9 shows that nodes on a grid are interconnected using some form of hardware Network Interface Card (NIC) and communications media. A functional software application may then send and receive messages via the NIC, by binding to an exposed socket or network service end-point.

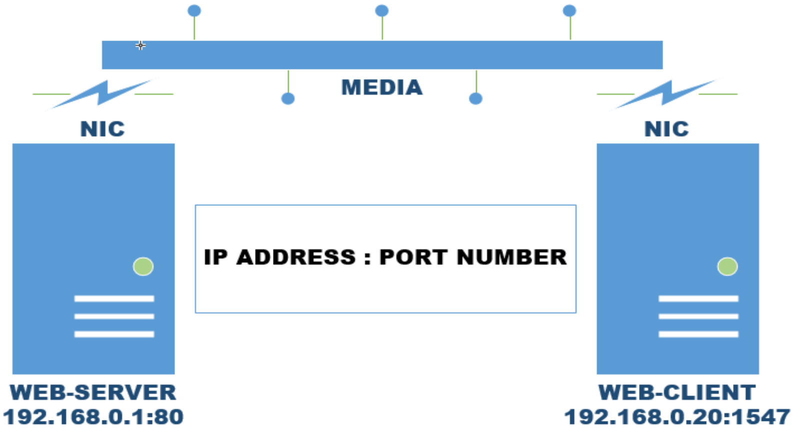


Figure 9 Simplified sockets

A socket fundamentally comprises an Internet Protocol (IP) address together with a logical port number (Jones & Ohlund, 2002; Kozierok, 2005; Mclean & Thomas, 2010). In rudimentary communication actualities, applications use derived socket information to fully qualify the source and destination of network traffic. The IP address portion, uniquely identifies the network node, whilst the port number associates an interpreting application to an enabling communication channel. A communicating sender application, offloads a message by targeting the socket of a destination node.

2.5.1 Network Packet Formation

A message is normally first *segmented* or fragmented for fit to the underlying network communication protocols in an encapsulation process known as *packet formation*, before being transmitted across the physical media as a network frame. In opposition, the destination node decapsulates the packet and reconstitutes the message for interpretation by a receiving application.

The predominant underlying protocols that facilitate network communications over Local Area Networks (LAN) and the Internet is currently the *TCP/IP protocol suite*: *IPv4 and IPv6* (Jones & Ohlund, 2002; Kozierok, 2005; Mclean & Thomas, 2010; Murray, et al., 2012; Bidgoli, 2016). The IP portion of the TCP/IP acronym pertains to the logical addressing, naming and routing functions of the protocol suite. The Transmission Control Protocol (TCP) portion of the acronym presentation, a minor contradiction in terms, as two or more contrasting transmission control protocols are in fact provided for. In clarification, the de facto TCP protocol is used to establish connection orientated or synchronous, error free communications between participants. Whilst the User Datagram Protocol (UDP) per example produces faster, error agnostic, asynchronous or connectionless communications. The important consideration as it pertains transmission control is that an application's information interchange needs could potentially be conversant of the selectable transmission protocol and by choice, leverage intrinsic network service characteristics beneficially.

The network frame size is another important influence in network communication efficiency (Romanow & Bailey, 2003; Regnier, et al., 2004; Kozierok, 2005; Murray, et al., 2012). An application's messages are typically segmented and encapsulated within a transmission control protocol. The encapsulation process generates normatively structured network packets, containing the sequenced message fragments. The maximum payload size of message fragments within an encapsulated IPv4 TCP/UDP packet, could theoretically approximate up to 64 Kilobytes (Kb). Initially, these potentially large packet size limits may conceivably be perceived as flexible and ample extents. The reality however is that network transmission sizes could have more complex parameter considerations beyond the transmission protocol choice. The interplay between the lower layers of the logical network model, could impose additional variables that may subvert network efficiency planning (Kozierok, 2005; Murray, et al., 2012). The network and data link layers of the logical network model requires mandatory additional utility overhead on a per layer basis. The overhead applied would reduce the amount of payload space a message fragment could occupy.

An effective strategy geared towards network efficiency, would therefore require insights into the underlying network capabilities beyond initial transmission protocol selection and configuration.

2.5.2 Packet Fragmentation Characteristics

The *Maximum Transmission Unit* (MTU) of a network frame is determined by use of restrictions enacted by logical or physical network devices along the transmission path (Romanow & Bailey, 2003; Regnier, et al., 2004; Kozierok, 2005; Murray, et al., 2012). The MTU is fundamentally the largest single message segment that can traverse a network without causing further fragmentation. The MTU size can be established programmatically at run-time for point to point communications, or alternatively automatically or manually configured per network device. The importance of the MTU value stems from its association with the underlying network capabilities. Regardless of TCP/UDP packet size programmatically or indirectly configured, an application message may potentially be further fragmented for fit inside the MTU restriction.

The configured size of the MTU value would have significant impact on the quality of network communications and the overall performance efficiencies sought (Romanow & Bailey, 2003; Regnier, et al., 2004; Kozierok, 2005; Murray, et al., 2012; Prakash, et al., 2013). The effects of the MTU size on the quality and performance of network communications are found to be vigorously documented in the peer reviewed literature. The consensus viewpoints expressing direct association towards quality and performance of network communications attributed to MTU size dynamics. The impacted areas of interest, highlighted as:

- Processing overhead (CPU, system, generic network and protocol)
- Network throughput and bandwidth utilization
- Network end-to-end latency
- Data serialization delay and jitter

A large MTU value could potentially alleviate network congestion, reduce protocol and system related processing overhead, whilst increasing network throughput (Regnier, et al., 2004; Murray, et al., 2012; Prakash, et al., 2013). Using large frames naturally requires less network packets to be formed and transmitted. The reduction in the number of transmissions should as consequence necessitate less processing overhead. A large frame also plausibly contains more application data payload per transmission, which ought to provide for better utilization of the available network bandwidth.

In opposition, a small MTU value should notionally reduce network latency and increase responsiveness (Regnier, et al., 2004; Murray, et al., 2012; Prakash, et al., 2013). Small frames would require less bandwidth and traverse internetwork devices in shorter amounts of time. Within a traditional socket environment problematic disadvantages could however manifest in either large or small MTU size choices.

The disadvantages that could possibly result from large MTU values comprises (Murray, et al., 2012): reduced error checking effectiveness, increased latency and inefficient packet queuing. The cyclic redundancy error checking mechanism's processing performance, could substantially degrade for frame sizes in excess of 12000 bytes. Packet prioritization also becomes problematic in especially low link speed environments. As larger frames take longer to serialize and transmit, delay and jitter may occur when prioritized packets are superseded in the queue. The possible drawbacks of small MTU sizes in comparison to large frames, are noted to be increased processing overhead and network congestion (Murray, et al., 2012). The more packets that require headers generated, the higher the processing burden to manage such. Equally, the propagation of additional packets could exacerbate network bottlenecking.

A deductive argument within the context of MTU frame sizes towards a balanced or mediated solution may be made. In lieu of advantages and disadvantages exhibited, this would indeed seem to be the prudent approach. Though to affect any such compromise for MTU frame sizes, it should be acknowledged that complete administrative control over the end to end network environment is needed (Kozierok, 2005; Murray, et al., 2012; Prakash, et al., 2013). As shown in Figure 10, the salient issue is that all homed and internetwork devices require MTU size compatibility. The lowest MTU configured from the network device aggregate, should not to be surpassed as inefficiency or dysfunction would result.

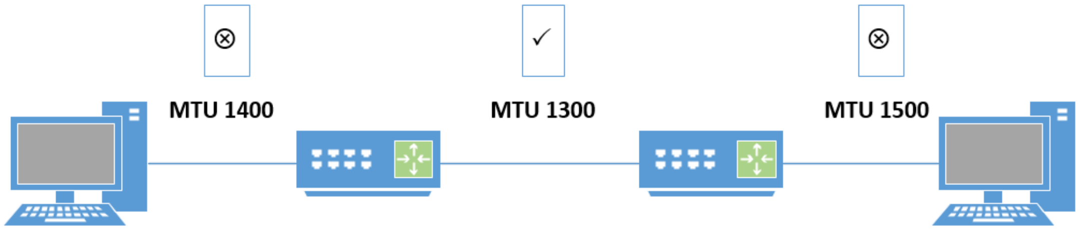


Figure 10 MTU path minimum size aggregation

Nevertheless, the use of large MTU frames, also popularly termed as *jumbo frames*, are known to substantially outperform legacy Ethernet based MTU implementations (Murray, et al., 2012; Prakash, et al., 2013). The term jumbo frame, broadly refers to frame sizes larger than the default Ethernet MTU of 1500 bytes. A recommended jumbo frame size approaching 9000 bytes is reported to be optimal for generic high throughput and performance communication requirements. The benefits of large frames, correspondingly hold true for jumbo frames. Notably, when supporting network device compatibility for jumbo frames are available, the historic disadvantages of large frames are frequently negated.

2.5.3 Traditional Socket Networking

The processing overhead, I/O bandwidth bottleneck and latency produced by traditional socket-based network communications, creates significant concerns for HPC or HTC environs (Speight, et al., 2000; Microsoft Corporation, 2001; Romanow & Bailey, 2003; Regnier, et al., 2004; Jin, et al., 2005; Zhang, et al., 2012; Prakash, et al., 2013). Primarily as it regards traditional socket based communications, grid nodes require OS kernel interactions by use of socket APIs. The use of any generic kernel API invocation, would predictably contribute CPU cycles to a software thread context. Yet when considering the use of kernel provided API socket calls, the resultant performance impacts could become especially pronounced (Microsoft Corporation, 2001; Jones & Ohlund, 2002; Romanow & Bailey, 2003; Microsoft Corporation, 2017). Dependend on the message workload, the application's socket configuration and interactions, the resultant CPU overhead from API socket invokes could degrade performance substantially. For the most part the functional application's use of the API message handling apparatus, could for instance cause undesirable processing stalls due to socket *blocking* operations (The WinSock Standard Group, 1997; Jones & Ohlund, 2002; Zhang, et al., 2012). A blocking event occurs when an associated process invoked by a socket API function call does not return until completion of the operation. Fundamentally, application processing can be halted or enter a quasi unresponsive state pending socket function completion.

An exacerbating factor of interest is the fact that host processing overhead associated with network I/O increases in high speed network environments (Speight, et al., 2000; Romanow & Bailey, 2003; Regnier, et al., 2004; Jin, et al., 2005; Zhang, et al., 2012). The higher the ratio of network link speed compared with internal system bandwidth, the more evident the I/O bottleneck. The combinatorial effects of overhead and bottlenecking increases overall latency, which would be undesirable in high performance or throughput communications.

In the interest of communications efficiency, when programming for traditional socket-based networking, at least *overlapped I/O with completion ports* (IOCP) are recommended (The WinSock Standard Group, 1997; Jones & Ohlund, 2002; Regnier, et al., 2004; Zhang, et al., 2012; Microsoft Corporation, 2017). When mapped onto hardware threads, the asynchronous nature of overlapped I/O can alleviate the effects of processing overhead associated with traditional network sockets. For all intents and purposes, an application's API socket calls return immediately, which allows user mode processing to continue in a non-blocking fashion. The socket operations initiated by the funtional application is completed by the OS kernel in the background. An application then interrogates message queues associated with a number of instanced sockets, known as I/O completion ports, to determine the status of socket operations.

2.5.4 Performance Networking

The varied combinations of supporting network hardware and media, together with OS kernel provisioning could considerably improve holistic network performance in comparison with traditional socket operations (Speight, et al., 2000; Microsoft Corporation, 2001; Romanow & Bailey, 2003; Liu, et al., 2004; Regnier, et al., 2004; Jin, et al., 2005; García-Dorado, et al., 2013; Prakash, et al., 2013; Ali, et al., 2014; Kalia, et al., 2016). It should be noted that the enabling solutions alluded to here, are in some instances programmatically transparent to a functional application's use of sockets. A performance enhanced, attuned and transparent solution, may therefore possibly not necessitate a rewrite or recompilation of the application. However it would be important to progressively elaborate, as well as dissect each attribute of a combinatorial solution in order to accurately establish applicability and compatibility.

The socket transparent NIC technologies of *interrupt moderation*, *checksum* and *large segment offload* as implemented in modern hardware, could lessen network related processing overhead whilst increasing throughput (Romanow & Bailey, 2003; Regnier, et al., 2004; García-Dorado, et al., 2013; Prakash, et al., 2013). The term interrupt moderation refers to the ability of a NIC to accumulate a number of incoming network packets before signalling once for CPU interaction, thereby liberating processing cycles. The word paring of checksum offloading, denoting the removal of responsibility for calculating TCP/IP error control checksums from the CPU. Instead, checksums are calculated using the NIC hardware. If large segment offload is supported, the NIC takes on the duties of processing application messages into segments and forming the corresponding TCP/IP headers. Large segment offloads could greatly increase egress throughput and reduce host processing linked with TCP/IP socket based network communications.

Further overhead reductions for traditional socket implimentations, can be gained from OS provisioned and improved socket API functions, that directly or indirectly implement *zero copy* features (Microsoft Corporation, 2001; Jones & Ohlund, 2002; Romanow & Bailey, 2003; Jin, et al., 2005; García-Dorado, et al., 2013). The switching between user and kernel execution contexts, would normatively necessitate data duplication. A data copy operation, innately accrues overhead by consuming memory bandwidth and processing cycles. A zero copy feature eases context switching overhead by providing a means to bidirectionally copy file data via sockets, without changing execution modes. In essence the kernel context is often bypassed, as data movement could occur between the memory spaces of the NIC and application directly.

2.5.5 Contemporary HPC and HTC Networking

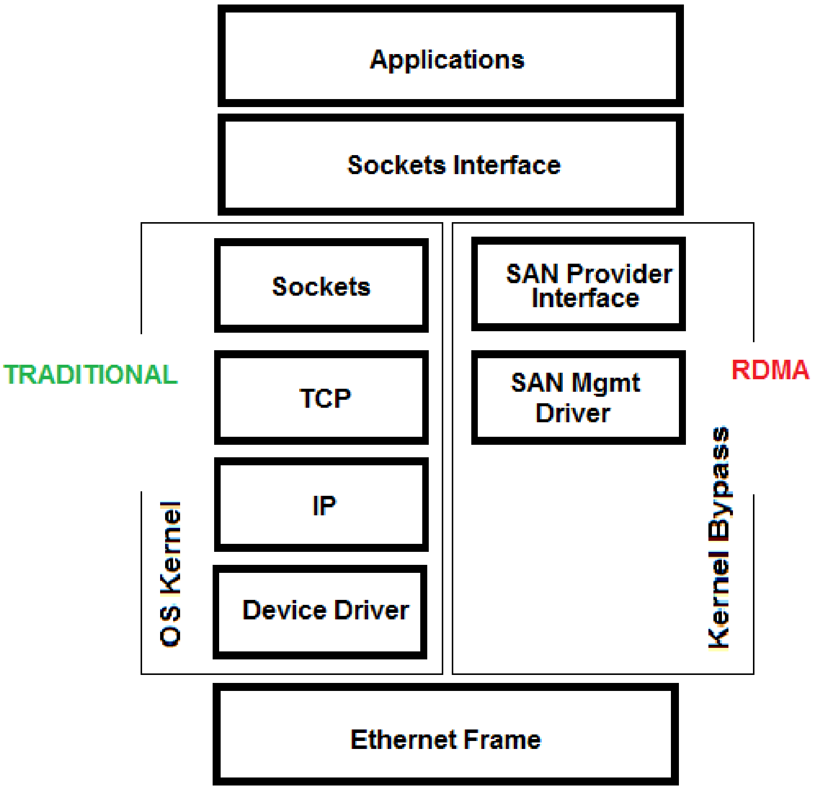


Figure 11 Network stack comparison
(adapted from Microsoft MSDN, 2001 and Jin et al, 2005)

The most widely purported combinatorial solution for high performance or throughput networking, which potentially provides best fit for HTC and HPC, implement forms of *Remote Direct Memory Access* (RDMA) by means of hardware infrastructure and platform provisioning (Microsoft Corporation, 2001; Romanow & Bailey, 2003; Liu, et al., 2004; Jin, et al., 2005; Ali, et al., 2014; Kalia, et al., 2016).

The network topology for RDMA solutions are constructed using specialized and often expensive hardware, which intrinsically expose high bandwidth and low latency characteristics. Of overarching significance however is that RDMA featured networks, provide the capability to directly access the memory of remote participant nodes. As shown in Figure 11, the mechanics of RDMA implement hardware accelerated versions of previously discussed features such as zero copy, large frame transmission and kernel bypass.

Compatible internetwork devices, node hardware, physical media and supporting OS kernels, frequently combine to form the holistic RDMA support solution. The result itself is often termed a *System Area Network* (SAN), referring to the vendor provided hardware and software platform features that may be exploited. Some popular examples of SAN networks include: iWARP (Internet Wide Area RDMA Protocol), RoCE (RDMA over Converged Ethernet) and InfiniBand.

2.5.6 Grid Formation and Fault Tolerance

The fundamental clustering of computing resources into a singly managed object for either HPC or HTC utility, may be presented as a two-tier network topology (Foster, et al., 2008; Shvachko, et al., 2010; White, 2012 ; Reyes-Ortiz, et al., 2015). A server node (i.e., a master, name node or resource manager) associates and interacts with multiple networked client nodes (i.e., workers, slaves or compute resources). The relationship has also been illustrated in Figure 12. The responsibilities of the server node encompass scheduling of client processing workloads and managing the integrity of the functional distributed computing platform.

In most HPC and HTC implementations, an optional data tier (i.e., a data node) as either a separate entity or imposed role can be added to improve data locality (Foster, et al., 2008; Shvachko, et al., 2010; White, 2012; Zhao, et al., 2014). Specifically for HTC platforms, the design could call for an implementation of a *Distributed File System* (DFS) to extend the data locality scenario for enchanced benefit (White, 2012). A DFS fundamentally constitutes a virtual file and directory hierarchy abstraction, or *namespace*. Each particular entry in the namespace in actuallity a logically mapped or aliased remote data storage location.

When established and appropriately configured onto a name node, the features provided by the DFS namespace may consequently be enhanced to include the replication of data and/or computation state between networked participants.

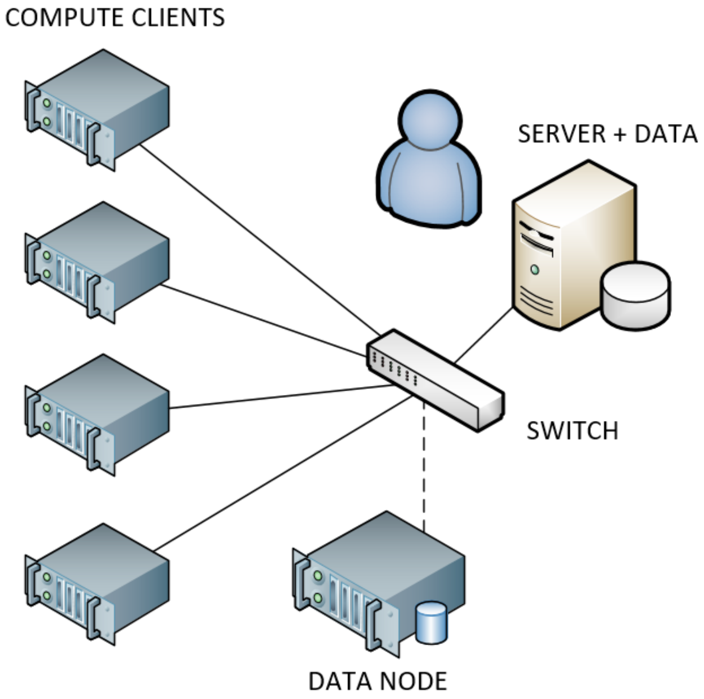


Figure 12 Elementary grid formation for HPC or HTC

The design of the DFS replication scheme, could per example enable desirable stability features such as *load-balancing* and *fail-over* between participants (Montero, et al., 2011; White, 2012). Should HTC participant nodes transition into an offline state for any given reason, the integrity of the computing environment could viably be maintained. The computation workload and data of the faulting nodes, would remain accessable by means of preconfigured replicants (i.e, fail-over, fault tolerance). Similarly, the faulting nodes' workload can be resheduled evenly among the remaining aggregate compute clients (i.e., load-balancing). The supplementary benefits of such DFS configurations innately easing the platform constraints of scalability and availibility.

The major differences in grid formation between HPC and HTC, concern the construction of the computation platform and the ability to deal with discreet node failure. The scheduled workloads of HPC platforms normally require static network topologies in order to function and therefore do not scale well (Ali, et al., 2014; Mantripragada, et al., 2015; Reyes-Ortiz, et al., 2015). Whereas in the case of HTC platforms, some topology dynamism is facilitated via inherent design (White, 2012; Reyes-Ortiz, et al., 2015). For HPC, node failure historically generates uncertainty in attaining intended outcomes, whereas HTC is semi tolerant of such failure.

2.6 Chapter Summary

This chapter started off by articulating the conceptual preparations and literature survey parameters employed in research material selection. Next, the historic roots and contemporary nature of technical debt phenomenon was investigated. Importantly, a link was made between technical debt and e-waste. The growing trend of organisational cloud and related services adoption, presented evidence towards reduced lifecycles for currently retained IT assets. The imminent loss of value as a result, is emphatically unattractive. To delay the burden of technical debt, the repurposing of assets that still conform to the organisational need is recommended.

The probable sources of future technical debt, are identified as platforms and infrastructure, supporting the Microsoft operating system. In this section and in previous deliberations, it was established that on demand computing power, by means of grid computing could pose as viable solution to the capacity and alignment problem. Specifically the grid applications of HPC and HTC, provide fit and exude the same value proposition characteristics of the cloud and modern data analytics. However, the nature of parallelism and heterogeneity in HPC and HTC is found to be problematic.

The chapter’s following inquiries, explore the parallelism and heterogeneity issues in an atomic fashion. A conceptual model that represents a computing platform as a grid participant, is dissected in an effort to understand underlying issues for HPC and HTC implementations. The resulting awareness engendered by the focused literature investigations, are to be instrumental in creating the generalizable design construct.

CHAPTER THREE
RESEARCH METHODOLOGY

3.1 Research Theory Development

The selection of research methodology should first and foremost be cognisant of the *ontological stance*. An ontology would speak to an inherent conviction of how reality would be constructed or experienced (Neuman, 2011; Bryman, et al., 2014). Notably the researcher's observational point of view and influence on the perceived reality should be garnered. Given an ontological decision, various more focused *epistemological avenues* are revealed. An epistemology refers to how scientific discourse and knowledge accumulation may be facilitated (Neuman, 2011; Bryman, et al., 2014). The outflow of the epistemology choice could indicate an ideal *methodological paradigm* to use. A research methodology describing the conceptual framework or pattern, together with a research process by which studies are conducted.

The holistic objective of the research exploration proposed needs to apprise on the practical applicability in design, as indicated by the expected contribution. Existing theory informs design, but during exploration using an aligned research methodology, potential emergent theory may result that could add academic value. As shown in the Figure 13, an adapted *Pasteur's Quadrant*, the pure applied research region is well suited to depict such undertaking. The quest for fundamental understanding is downgraded in favour of considerations regarding value in use, whilst inherently not excluding theory enrichment opportunities. The Design Science Research paradigm is known to be applicable in research environments that propose to develop new designs, software artefacts and instantiations (Fischer, 2011; Prat, et al., 2014).

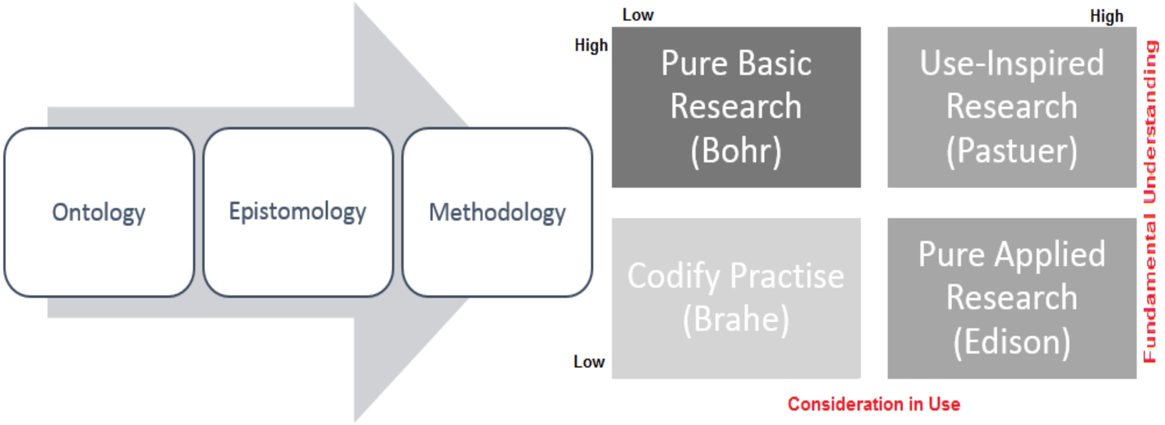

Ontology
Epistomology
Methodology
Low
High
High
Pure Basic Research (Bohr)
Use-Inspired Research (Pastuer)
Codify Practise (Brahe)
Pure Applied Research (Edison)
Low
Consideration in Use
Fundamental Understanding

Figure 13 Research and theory development
(an adapted Pasteur's Quadrant as originally proposed by Donald Stokes, 1997)

3.2 Design Science Research

Information Systems (IS) are implemented to enable some arrangement to enhance effectiveness and/or efficiency. The computerized IS, could arguably be described as interrelated and in most cases interdependent artificial constructs of engineering (Brooks, 1996; Simon, 1996).

Design-Science Research (DSR) has proven a valid conceptual, execution and evaluation framework for the study of IS, which has been gaining larger traction within the scientific information technology research community (Fischer, 2011; Prat, et al., 2014). DSR's origins can be traced to the latter half of the 1980's, with rudimentary maturity in research application being witnessed after the mid-1990's. Primarily DSR in the legacy setting was used as an emergent pattern for engineering the artificial (Simon, 1996). When discussing DSR, the use of the term *processes* clarifies and circumscribes the interrelated as well as interdependent nature of IS constructs. Correspondingly the term *artefacts* in the conceptual framework of IS, delineates the artificially engineered; thus targeted functional computer programs together with their means of operation (Hevner, et al., 2004).

The DSR archetype endeavours to create or expand capabilities via innovation (Hevner, et al., 2004). The target area of application for such innovation is real world human or organizational environments. Typically, such environments could benefit from innovative application artefacts as solutions to given problem domains. A clear understanding of the given problem domain facilitates an understanding of the performance characteristics of an artefact and whether it indeed alleviates the identified problem (Hevner, et al., 2004). The outcome of DSR endeavours are artefacts, instantiations, methods and models that add value in practice (Hevner, et al., 2004; Kuechler & Vaishnavi, 2008; Prat, et al., 2014). Fundamentally DSR provides a basis in which artefacts can be developed whilst also imparting prescriptions as to the process of such artefact creation and eventual use.

The context of the perceived problem provides input into the initial design process, which in turn outputs a possible *design artefact* as solution (Hevner, et al., 2004; Kuechler & Vaishnavi, 2008; Fischer, 2011; Prat, et al., 2014). Artefact construction may in cases, precede the knowledge of why and in what manner it functions (Simon, 1996; Gregor & Jones, 2007). Importantly the candidate design artefact is evaluated as to its suitability for solving the problem. The evaluation process in turn expands the understanding of the problem.

Similar to classic transformation models and *Soft Systems Modelling,* timeless characteristics of input, process, output, feedback and control are created, in turn forming an iterative loop of *build-and-evaluate* until the final design artefact is generated (Checkland, 1985; Hevner, et al., 2004; Kuechler & Vaishnavi, 2008; Prat, et al., 2014).

According to Hevner et al. (2004) "*Truth informs design and utility informs theory*". As social-science research endeavours to justify, develop or predict behavioural truth; design science seeks to establish utility that informs such hereto unknown truth (Hevner, et al., 2004; Fischer, 2011). DSR artefacts are seldom all-inclusive information systems that make it into practice, however during the endeavour of research the ideal solution to the problem domain becomes manifest, which enhances and adds to the knowledge base (Hevner, et al., 2004).

Design-Science as research paradigm, is suited for addressing historically difficult IS problems (Hevner, et al., 2004). Characteristics of such difficult IS problems include: instability of requirements, integration complexity, fluid process or design environments and dependency of successful outcomes on human interaction (Brooks, 1996; Hevner, et al., 2004). These difficulties as described, are assuredly not unique to the field of IS. Per illustration, nearly all management knowledge areas suffer the same or similar difficulties and are therefore popular topics for books as well as field research (Ward & Peppard, 2002; Schiesser, 2010; Gido & Clements, 2014). And yet, a distinguishing factor amongst others is that DSR provides non-mandatory guidelines, shown in Table 7, which specifically alleviate complications to IS artefact creation.

Table 7 Seven guidelines of design science research
(adapted from Hevner, et al., 2004)

1	Construct purposeful utilitarian artefacts
2	Ensure alignment with the problem domain
3	Evaluate artefacts thoroughly for intended functionality
4	Innovation is key for heretofore unsolved- or effectiveness/efficiency problems
5	Rigorously defined, formally presented, coherent and internally consistent
6	Numerous solutions to the problem are possible, technique optimal choice criteria
7	Communicate effectively to the mixed stakeholder audience of the problem domain

Rigor ultimately augments the relevance and importance of the DSR endeavour (Hevner, et al., 2004; Ostrowski & Helfert, 2012; Venable, et al., 2016). Notably guideline 5, '*rigorously defined, formally presented, coherent and internally consistent'* is then ultimately the major difference between DSR and other comparable paradigms. Interestingly this supposition by Hevner, et al. (2004) differentiating DSR was initially disputed, but later found to have achieved academic validity (Fischer, 2011; Prat, et al., 2014). The contemporary seminal scholars whom are contributing to DSR, consequently and consistently reiterate broad agreement with Hevner, et al. (2004) guidelines. The DSR author contributions and research standpoints, in the reviewed publications after Hevner, et al.'s (2004) initial work, then frequently predominate around practical implementation or interpretation of these guidelines.

3.2.1 Practical Application

A paper by Gregor & Jones (2007) emphasises that goal and scope, together with constructs of theory be apparent in the DSR descriptive. Comparison of theory with similar prior theories when indicating goal and scope of research should provide metrics (Gregor & Jones, 2007). The quantification of such is important, for it allows a means to determine the contribution of research to the body of knowledge. Investigating similar theories and how the research theory under review was derived, reveals the amount of value innate (Gregor & Jones, 2007). Accordingly, the undesirable tendency in academic research to generate additional terms and theories, which duplicate existing knowledge as well as technologies may be curbed. The influential DSR authors Peffers, Tuunanen, Rothenberger & Chatterjee bolsters the DSR methodology in late 2007. Highly acclaimed, the published work dramatically increases DSR acceptance within the IS research community. Of significance is that Peffers, et al. (2007) provides the research practitioner with a consensus conceptual methodology framework and toolset by which to implement DSR consistently. The procedures, practices and principals for DSR as described by Peffers, et al. (2007) may be used to target all-inclusive IS research actions. The authors highlight three key objectives in creating the methodology enhancements for DSR namely: nominal research process model (see Figure 14), terminology consistent in literature and a mental model for evaluating IS research. Importantly the activities described in the DSR process model have direct fit to Hevner, et al.'s (2004) guidelines, but have demonstrated practical application value. Additionally, the mental model for DSR provides insights into gauging the research worth, during and post development (Peffers, et al., 2007). The DSR mental model could for example serve as instrument in determining whether envisioned research objectives are being achieved, or exhibit evidence towards enhancement of the research body of knowledge.

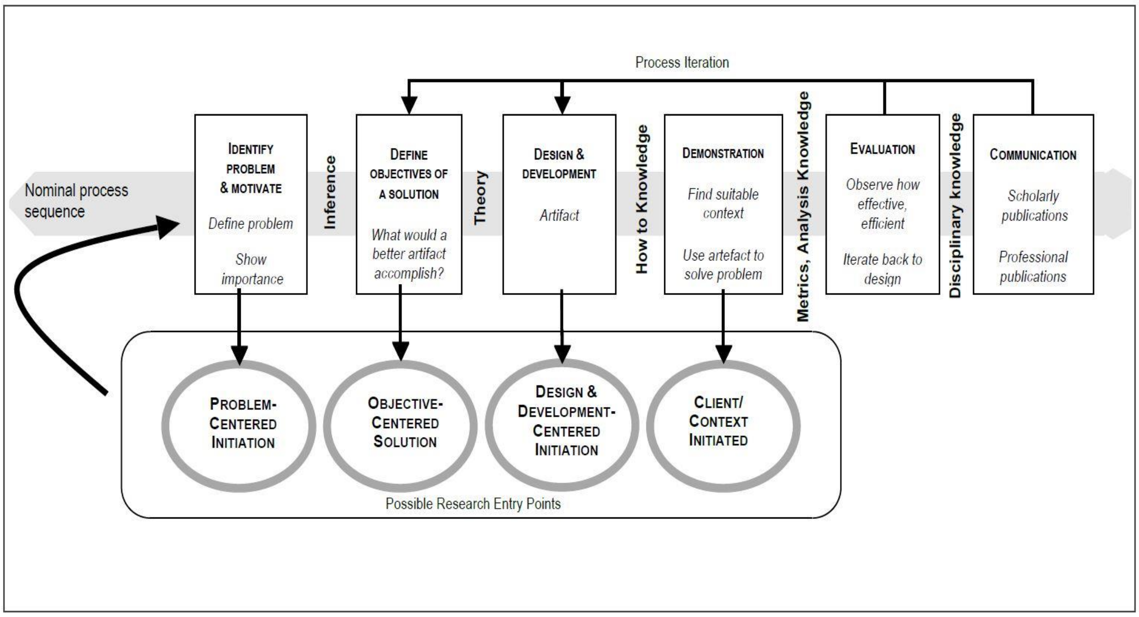

Process Iteration
Nominal process sequence
IDENTIFY PROBLEM & MOTIVATE
Define problem
Show importance
Inference
DEFINE OBJECTIVES OF A SOLUTION
What would a better artifact accomplish?
Theory
DESIGN & DEVELOPMENT
Artifact
How to Knowledge
DEMONSTRATION
Find suitable context
Use artefact to solve problem
Metrics, Analysis Knowledge
EVALUATION
Observe how effective, efficient
Iterate back to design
Disciplinary knowledge
COMMUNICATION
Scholarly publications
Professional publications
PROBLEM-CENTERED INITIATION
OBJECTIVE-CENTERED SOLUTION
DESIGN & DEVELOPMENT-CENTERED INITIATION
CLIENT/ CONTEXT INITIATED
Possible Research Entry Points

Figure 14 Design science research nominal process model
(adapted from Peffers, et al., 2007)

The academics Kuechler & Vaishnavi, (2008) recommend that narratives underpin and are applied to all abstracted models. Metaphors that tell or encourage a story, yield better results to conceptually focus stakeholders whilst stimulating goal orientated thinking. Pointedly, "*grammatical element salience in conceptual modelling*" enhances all of Hevner, et al.'s (2004) other guidelines (Kuechler & Vaishnavi, 2008). Secondary benefits of this approach are that it provides for more accurate, common, persistent and communicable understanding of the problem domain. Such understanding is then extended to encompass the artefact design, development and evaluation process steps.

3.2.2 Strategy, Outcome and Process

The DSR contributor Juhani Iivari, reiterates and promotes cognisance of the two contrasting research strategy choices particular to information systems. Expressively the choice regarding strategy, could have very real implications for the context, outcomes, process and resource requirements of a DSR undertaking (Iivari, 2015). Consequently, Iivari clarifies and differentiates the two DSR strategies that target information systems, along sixteen apportioned dimensions. Guided by these dimensions of a strategy choice, the researcher should be able to holistically comprehend the research environment, maximize the research process for possible benefits, avoid pitfalls and add focus to the potential research contribution sought.

In the first identified research strategy choice (i.e., Strategy 1), a case for generalizability of solution to a perceived problem domain is created by building meta-artefacts, using overwhelmingly objective and deductive patterns (Iivari, 2015). The resultant meta-artefacts, may or may not eventually become instantiations. The second DSR strategy (i.e., Strategy 2) differs by attempting generalizability of solution through the development and subsequent reflection on, real-world system implementations (Iivari, 2015). The dominant deliberations around Strategy 1 or 2, should therefore *contextualize* whether the envisioned research requires interaction with a perceived client entity and their specific problem. The fundamental divergence on the DSR strategy, regards the concept of general problem domains versus entity conveyed, specific or so-called concrete problem domains.

Strategy 1 would be conversant of multiple uncertainties surrounding the problem in practice, how the solution may be formulated and what the eventual research contribution will be (Iivari, 2015). For Strategy 2 the problem presents with immediate complexity dimensions and an involved client entity. The path towards the solution in this strategy is also unclear, but considerable uncertainty regarding the eventual research contribution is generated (Iivari, 2015). It could be said that both strategies have central, but dissimilar degrees of uncertainty surrounding the problem and direction toward possible solutions. Nonetheless of more substantial and imminent concern for the research practitioner would be, whether perceptible research contributions could ultimately be realized by means of the DSR strategy choice.

Once furnished with the DSR strategy selection, the avenues to likely research *outcomes* may be generated (Iivari, 2015):

- Strategy 1 - meta-artefacts (optionally instanced), proof of concept evaluations, reasoned design that exhibits innovation and exposes possible practical applicability
- Strategy 2 – instanced artefacts for a specific problem (possibly meta-artefacts as DSR contribution), evaluations that concern real-world systems that depict innovation by design, proof of practical applicability

The outcomes of both strategies, engenders firm onus onto the researcher to uncover value and build legitimacy for claimed contribution to the body of knowledge. The theme of reasoned design that includes innovation is also reflected in either strategy outcome. Key variances are however explicit for the outcomes of practical applicability and evaluation targeting, as a natural outflow of causal strategy contextualization.

Investigating the research *processes* or methods attributed to the DSR strategy choice, demonstrates similar disparity along the same motif (Iivari, 2015):

- Strategy 1 – iteratively constructed meta-artefact as generic solution, empirical artefact evaluation, generalization stemming from and informed by the problem statement
- Strategy 2 – experiences from a specific solution in practice, action-based research as intervention, constructions around concepts may yield meta-artefacts (elective empirical evaluation), generalization stemming from and informed by design patterns for a class of problem

For Strategy 1 the process is driven by means of cyclic versioning of candidate solution meta-artefacts, which should expose universal fit to the problem domain (Iivari, 2015). The major process activities in the strategy, are centred on building and empirically evaluating the artefact as an evolutionary progression. Seeking generalization is also noted as fundamental in the applied process.

The Strategy 2 process contrasts by using experiential knowledge of solving a problem in practice, as input into meta-artefact design patterns that may suggest generalizability of solution for a specific class of problem (Iivari, 2015). Again, the salient theme indicates that the researcher is immersed in solving a specific client problem. Studies conducted during solution creation could uncover design principles that have generic applicability for a class of problem. The design patterns and their generalizability rational are revealed or inspired during the client's system development process.

The final implication of DSR strategy choice for information systems, regards *resource requirements*. The resource needs for Strategy 2 is expectantly greater than its counterpart (Iivari, 2015):

- Strategy 1 – primary problem domain expertise (optionally sub-domain specialists), research team comprises single individual or small team, schedule and cost of research varies greatly dependant on ambition
- Strategy 2 – necessary client involvement, larger research teams (optional interdisciplinary teams), longer schedule periods and cost prohibitive

3.2.3 Artefact Evaluation Criteria and Methods

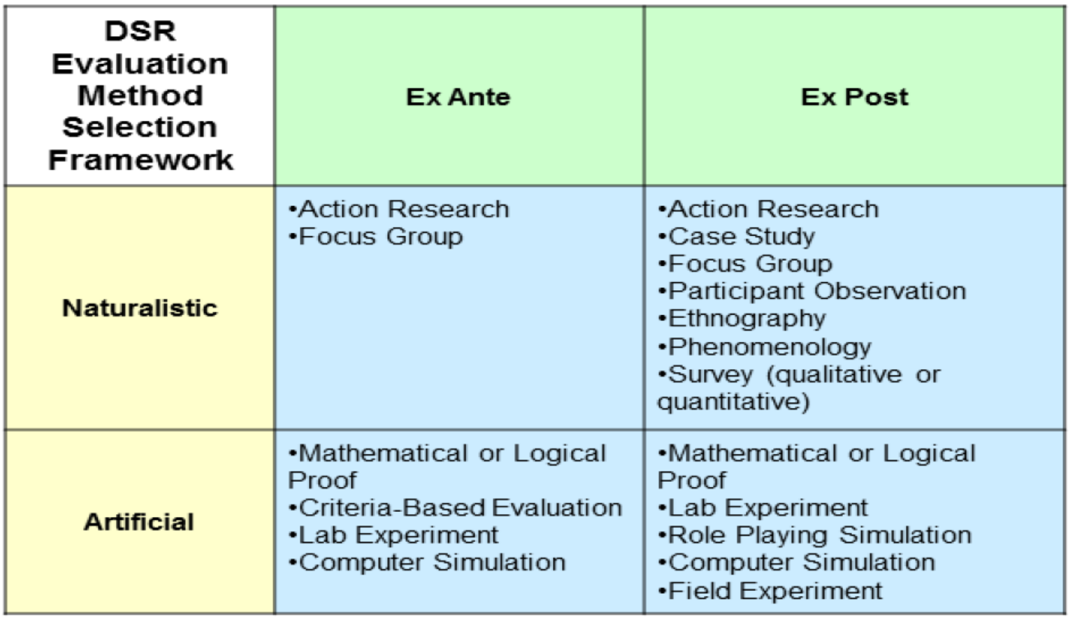
The representation of criteria and methods, by which DSR artefacts are vetted would be important in establishing the measure of research rigor and contribution (Hevner, et al., 2004; Peffers, et al., 2007; Pries-Heje, et al., 2008; Ostrowski & Helfert, 2012; Peffers, et al., 2012; Venable, et al., 2012; Prat, et al., 2014; Venable, et al., 2016). An information systems artefact, may certainly be evaluated using any number of criteria within a given context. Nevertheless the criteria and methods selected, are required to pass research community muster. In remedy to potential difficulties in establishing artefact evaluation validity, Hevner, et al. (2004) proposes five classes of evaluation methods as shown in Table 8. Of significance for the research practitioner is that the applicable methods within each class, has consensus authority in the research knowledge base. In abridged terms the design artefact's context is matched with the appropriate method(s) of evaluation.

Table 8 Design evaluation methods
(adapted from Hevner, et al. 2004)

Evaluation Class	Appropriate Method
Observational	Field Study Case Study
Analytical	Dynamic Analysis Optimization Architecture Analysis Static Analysis
Experimental	Controlled Experiment Simulation
Testing	Structural Testing Functional Testing
Descriptive	Scenarios Informed Argument

Also be reminded, that within the Peffers, et al., (2007) DSR process model the evaluation activity is conceived in two stages (Peffers, et al., 2007; Ostrowski & Helfert, 2012). The '*demonstration'* action, illustrates that the indented purpose of the artefact is feasibly achieved within at least a single context. Whereas the '*evaluation'* action, elaborates on the applicability of the artefact towards solving a problem. The appropriate method and criteria of evaluation, therefore requires further situational awareness regarding the current DSR process stage.

The artefact evaluation situational context, as depicted in Figure 15, can be described as either '*artificial'* or '*naturalistic'* (Pries-Heje, et al., 2008; Venable, et al., 2012). The scope or dimensions of the research strategy being the major driver of the artefact's evaluation strategy and method. An *ex ante* (i.e., prior to artefact creation) and *ex post* (i.e., after artefact creation) perspective can be used to codify the evaluation strategy and provide appropriate methods for evaluation (Pries-Heje, et al., 2008; Venable, et al., 2012). Additional factors of goal, condition and constraint can also be used as contextual inputs into the evaluation strategy and method choices (Venable, et al., 2012).



DSR Evaluation Method Selection Framework	Ex Ante	Ex Post
Naturalistic	•Action Research •Focus Group	•Action Research •Case Study •Focus Group •Participant Observation •Ethnography •Phenomenology •Survey (qualitative or quantitative)
Artificial	•Mathematical or Logical Proof •Criteria-Based Evaluation •Lab Experiment •Computer Simulation	•Mathematical or Logical Proof •Lab Experiment •Role Playing Simulation •Computer Simulation •Field Experiment

Figure 15 Selecting DSR evaluation methods
(adapted from Venable, et al., 2012)

The DSR framework known as FEDS (Framework for Evaluation in Design Science Research), is the de facto vehicle for contemporary artefact evaluation strategy selection. The FEDS framework, as shown in Figure 16, reveals evaluation strategy agendas for DSR artefacts, beyond the initial paradigm of naturalistic or artificial evaluations. Prominently, the functional purpose of the artefact's evaluation is considered to have weightings of formative and summative extents (Venable, et al., 2016). The proportions of summative evaluations would determine the magnitude of efficacy or matching outcomes to expectations. The scope of formative evaluations, endeavouring to increase the process or efficiency by which outcomes are achieved. When the dimensions regarding the paradigm of evaluation, counter to the functional purpose of evaluation is accordingly formed, the evaluation strategies are said to be (Venable, et al., 2016): Quick & Simple, Human Risk & Effectiveness, Technical Risk & Efficacy and Purely Technical.

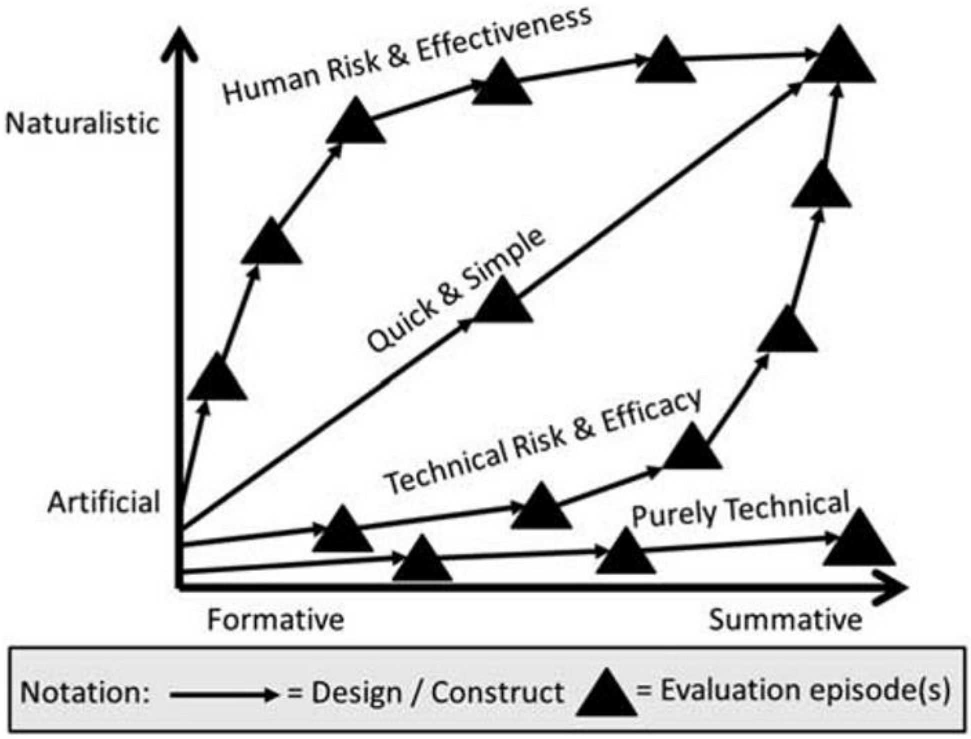

Naturalistic
Artificial
Formative
Summative
Human Risk & Effectiveness
Quick & Simple
Technical Risk & Efficacy
Purely Technical
Notation: = Design / Construct = Evaluation episode(s)

Figure 16 The FEDS artefact evaluation framework
(adapted from Venable, et al., 2016)

The characterization of evaluation strategies by use of the FEDS framework, explaining the idealized DSR evaluation process in terms of "*When to evaluate, for what purpose, and how*" (Venable, et al., 2016).

The Human Risk and Effectiveness evaluation strategy is well suited to the naturalistic paradigm. The strategy choice is enabled by research settings that display fundamentals of rigor and utility sought over longer periods of time, where research costs are perceived to be low and/or major social risk could manifest (Venable, et al., 2016). The Human Risk and Effectiveness strategy applies multiple episodes of meticulously conducted, typically naturalistic formative evaluations, which culminate in naturalistic summative evaluations. The effectiveness of the DSR artefact in providing long term utility or benefit for a client audience is said to be of significance.

The evaluation strategy as a DSR case for Quick and Simple, comprises a diminutive design endeavour, which presents with low technical and social risk (Venable, et al., 2016). Suited for more naturalistic problem domains, the evaluation is initially formative but develops rapidly towards a naturalistic summative effort. The Quick and Simple approach is then also pigeon-holed by few evaluation episodes and a low research cost environment.

The Technical Risk and Efficacy strategy is used primarily for evaluations of the artificial. Numerous and iterative artificial formative evaluations, progress towards summative evaluations that rigorously expose innate value of the design artefact. A final summative evaluation does however normally include naturalistic deliberations of the design artefact in use. The research environmental features that fit the strategy, present as having one or more of the following criteria (Venable, et al., 2016):

- The design endeavour's risk profile is predominantly technically orientated
- It would be impractical or incur high research costs, if conducted within a social setting
- The effectiveness of the artefact's utility or benefit, needs to be intrinsic and not necessarily reliant on social interactions

Lastly, the Purely Technical evaluation strategy selection is derived by the research circumstances where no social aspects are involved and/or where design artefacts are for future consideration (Venable, et al., 2016). The use of naturalistic evaluations are therefore irrelevant, only purely artificial formative and artificial summative evaluations are conducted.

The artefact evaluation strategy choice itself, is actioned through use of a FEDS four step process. Each step in the process engendering efficacy, rigor, goal orientation and environmental focus for the discreet evaluation activity. In telling contrast to other frameworks, Venable, et al., (2016) furthermore delivers evidence of risk reduction, effectiveness and efficiency in practical application.

The FEDS evaluation design process steps, are provided as (Venable, et al., 2016):

1. Clarify the goals of artefact evaluation
2. Select the appropriate artefact evaluation strategy or strategies
3. Conclude which properties of the artefact to evaluate
4. Plan the individual artefact evaluation episodes

The first step in evaluation design process establishes rigor, addresses ethics, reduces risk and uncertainty, whilst balancing efficiency against other goals (Venable, et al., 2016). The aspects of rigor include two key perspectives of artefact instantiation. In an artificial paradigm, the observed measure of the artefact should not be influenced by external factors. Whereas in a naturalistic setting, the effectiveness of the instantiation needs to be measured in the real-world environment. The application of ethics surrounding any evaluation, must endeavour to perpetually limit potential harm befalling stakeholders.

A proper evaluation design should also acknowledge social or technical risks and seek reduction of uncertainty. To achieve such risk and uncertainty reduction, it is recommended that formative evaluations be scheduled at the earliest possible stage. Meanwhile the efficacy planning context of the evaluation considers how research costs and resources may be prudently spent.

The second stage of the evaluation design process considers and selects an appropriate FEDS strategy dependant on the '*why, when, and how*' of the evaluation (Venable, et al., 2016). The characterization of the research environment, as it concerns the risks and constraints of the envisioned design artefact provides the primary direction of choice.

The FEDS evaluation design process's third step, determines the detailed properties of the artefact instantiation that will be the subject of evaluation (Venable, et al., 2016). The properties exhibited by the artefact, would present with unique features that link and frame the situational design goals. The artefact properties under review affords rational and justification towards the selection of scientifically valid evaluation methods.

The last step in the evaluation design process, prioritizes and schedules the evaluation episodes within the research environmental constraints (Venable, et al., 2016). The number and type of evaluations are notably contextualized dependant on resource availability. An artefact evaluation episode involves such factors as the time of evaluation, what to evaluate, resource requirements, method of evaluation and responsible party assigned.

3.3 Research Design

3.3.1 Context, Strategy and Intended Outcome

The DSR *Strategy 1* context and process descriptive for information system artefacts, accurately maps the intended research action. The research effort is primarily geared towards the design and instantiation of a utilitarian software artefact. A real-world client or customer will be absent from the research environment.

There is a need for objective and deductive build patterns to be persistently used in the design endeavour. The design artefact will be iteratively built and empirically evaluated. It is projected that the generalizability of the research contribution, would stem from the problem statement and the efficacy of the artefact as solution. The solution would need to expose innovation and provide indications of practical applicability.

The Hevner, et al. (2004) DSR guidelines and Peffers, et al., (2007) DSR nominal process model, will be applied in framing the research undertaking. The theory and practical application considerations of Gregor & Jones (2007), as well as Kuechler & Vaishnavi (2008) and Iivari (2015) can provide internal focus and feedback in attaining research value.

The design artefact evaluation strategy, criteria and methods will be conversant of the DSR evaluation method selection framework of Pries-Heje, et al., (2008) and Venable, et al., (2012). The evaluation strategy then also subjected to the FEDS subordinate evaluation strategy and four-step process model by Venable, et al., (2016).

3.3.2 Research Method

The envisioned design artefact and research environment, is conducive to the FEDS *'Purely Technical'* and *'Artificial'* evaluation model. Pointedly, the proposed design solution is future problem domain orientated and no social actors are required. Only quantitative methods such as mathematical or logical proofs, together with criteria-based evaluations are employed.

The bounded constraints of design and evaluation of the artefact, is informed by rationale established in the literature surveyed. The numerical measurement and reporting of functional extents, exposed by the instantiated artefact's system dimensions are of interest. The evaluations are conducted as multi episode activities within a laboratory setting.

3.3.3 Metrics and Analysis

The research units of analysis comprise the efficiency and efficacy levels of the design artefact. The usefulness and performance of the design artefact is therefore to provide a measure in determining overall applicability as potential solution. The units of observation contain the hardware, software and network components of discreet runtime instantiations. The data points as metrics of system extents, are to be gathered using only industry and academically scrutinized tools or techniques. The metric types in data collection across instantiations, would include varying physical or logical dimensions of effectiveness and efficiency. The presentation of quantitative research data takes the form criteria comparison tables, graph plots, mathematical and statistical schemes.

3.3.4 Validity

The research undertaking needs to make comparisons of theory with similar prior theories. The goal and scope of the research narrative should therefore provide for proportions, whereby the contribution of research to the body of knowledge may be measured. Building and exposing validity is intended to be a fundamental component of the research model.

3.3.5 Research Resources

The research case is built and informed by the architecture and operational environments as identified by subsequent research questions. Cognizance is taken of environmental characteristics that promote heterogeneous grid computing realities. A grid participant blend of virtualization, mobile, generic server and desktop settings is expected. Scalable academic computer laboratories located in the Western Cape, South Africa are the intended proving grounds for the envisioned design artefacts.

3.3.5.1 Code Development

To reduce the multitude of difficulties caused by abstraction and heterogeneity in current HPC or HTC platforms, the software design artefact is developed in assembly language. The initial motivation is that a reduced application stack can be generated by use of assembly language. The compiled executable artefact, consequently is not reliant on additional or intermediate run-time libraries and interpreters. A major additional motivation for the use of assembly language, is the requirement of direct access to hardware configurations as identified in the research. Furthermore, assembly language programming enables numerous optimization opportunities via unrestrained instruction set support.

3.3.5.2 Artefact Development Platform

The following platform was used as development and primary data gathering source: ASUS laptop Model: X554L, Windows 8.1 (x64) Non-Domain Bound, UASM (The Unified Assembler) fork of the Watcom assembler, MASM32 SDK libraries and macros, RADASM & Easy Code visual assembly integrated development environment, OllyDbg 2.01 (x86/x64) debugger, CodeXL 2.4 version Win 2.4.45, Intel Parallel Studio XE Cluster Edition for Windows 2018 (initial release), MS Windows ADK: Performance Recorder, MS Visio/Office 365 Pro, HxD – Hexeditor version 1.7.7.0. The laboratory environment for artefact instantiation and testing has been detailed in Table 9.

Table 9 Laboratory Environment

24	HP ProOne All-in-One 600, Intel Core i5-4590S 3 GHz 4 Core/Threads, 8GB RAM, 200 GB HDD, Microsoft Windows 10 Enterprise (x64) Build 16299, Domain Bound, MS Hyper V ver.10.0.16299.15, Intel I217-LM NIC in Full-Duplex 100Mbps, Average Passive Load: CPU 7%, RAM 46 %
1	HP ProCurve 2650 L3 Switch 48-port x 10/100 2 x SFP + 2 x 10/100/1000 Twisted Pair CAT 5E Network Cabling using TIA 568B wiring schema
1	Hyper V Virtual Machine, 2 x CPU, 2GB RAM, 40GB HDD, Windows 2012 R2 Server Standard Edition Build 9600, Non-Domain Bound
1	Hyper V Virtual Machine, 2 x CPU, 2GB RAM, 40GB HDD, Windows 8.1 Enterprise Build 9600, Non-Domain Bound
1	Hyper V Virtual Machine, 2 x CPU, 2GB RAM, 40GB HDD, Windows XP SP3 (x32), Non-Domain Bound
1	Hyper V Virtual Machine, 2 x CPU, 2GB RAM, 40GB HDD, Linux OpenSUSE Leap 42.3 DVD Edition, x86_64, Non-Domain Bound

3.4 The Design Cycle

It has been established that a central activity of DSR, concerns the design and study of a meta-artefact within a problem context. The interactions of the design artefact and the problem context should observably advance by means of outcome the utility realized. The depiction of the design effort would therefore play a vital role in mapping the design artefact to the problem context. In order to structure a resilient DSR design descriptive, the proposed outline by Roel Wieringa, shown in Figure 17, known as the *Engineering cycle* or *regulative cycle* is applied.

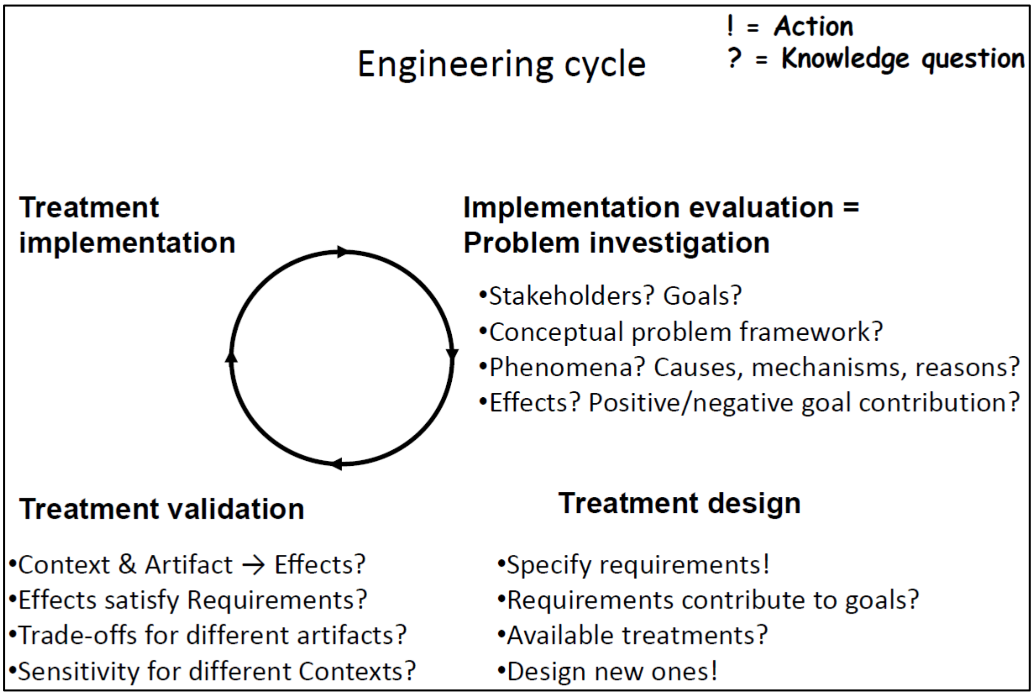

Engineering cycle
! = Action
? = Knowledge question
Treatment implementation
Implementation evaluation = Problem investigation
•Stakeholders? Goals?
•Conceptual problem framework?
•Phenomena? Causes, mechanisms, reasons?
•Effects? Positive/negative goal contribution?
Treatment validation
•Context & Artifact → Effects?
•Effects satisfy Requirements?
•Trade-offs for different artifacts?
•Sensitivity for different Contexts?
Treatment design
•Specify requirements!
•Requirements contribute to goals?
•Available treatments?
•Design new ones!

Figure 17 DSR engineering cycle
(adapted from Wieringa, 2016)

The Engineering cycle initiates by means of ‘*Problem investigation’* and ‘*Implementation evaluation’* (Wieringa, 2009; 2016). The stakeholders and their goals, as well as the environment, describes a conceptual problem framework. Particularly the problem framework, could potentially benefit by means of new technology introduction. The current effects experienced, their causes and mechanisms within the problem context are of interest. The magnitude of the effects on stakeholder goals, are an expression of the problem context importance. The core values in the problem investigation stage, therefore regards the description and diagnosis of the problem context (Wieringa, 2009).

As potential solution, the eventual design artefact could realistically be imperfect or generate additional problems within the problem context (Wieringa, 2009; 2016). A design solution is consequently understood to be either a singular or possible '*treatment'* to the problem context. The solution design stage of the Engineering cycle, is then aptly labelled '*Treatment design'*. A design is specified that argues and documents the distinct design decisions (Wieringa, 2009; 2016). The requirements of within the design specification, needs to have clear problem context and goal counterparts. Importantly, relevant existing treatments are deliberated and new designs considered.

The '*Treatment validation*' stage justifies the goal contribution before artefact instantiation (Wieringa, 2009; 2016). Essentially the design artefact's predicted effects are conceived to satisfy the stakeholder requirements as knowledge tasks. The design requirements themselves having narrow fit with the original research questions. The internal and external validity of the design, together with design trade-off studies, should create norms of inherent design value. The internal validity step, attempts to satisfy the design against the criteria set out within the problem investigation (Wieringa, 2009). While trade-offs or alterations in the design are considered, in determining whether problem criteria would remain satisfied. The external validity or sensitivity context step, investigates whether the design could meet similar or same criteria within an altered problem context (Wieringa, 2009; 2016). Feasibly then, the design's sensitivity within an altered context, would infer generalizability of a particular treatment.

The '*Treatment implementation'* stage, concludes the Engineering cycle. The prototype construction of the design is undertaken (Wieringa, 2016). The design is executed and may then be assessed within discretionary evaluation sequences.

3.5 Chapter Summary

To begin with, the chapter modelled generic research theory development as a process. Afterwards the discussions build a case for the Design Science Research paradigm, as a pertinent methodology to conduct research. The importance of the DSR research context, strategy, method and outcome for information systems was deliberated. The written accounts of strategy frameworks, techniques and models for DSR artefact evaluation, provide additional value in research practice. Lastly the research action's design, the artefact's development and execution environment were detailed.

CHAPTER FOUR
ARTEFACT DESIGN

4.1 The Technical Debt Context

4.1.1 Problem Investigation

The root causes and effects of technical debt, are acknowledged to be multi-dimensional in literature. However, a major contributing factor for increased technical debt in the near future, may be cloud and related services adoption. The reasons for organisational cloud adoption, regard subjects such as on-demand capacity generation, leveraging potential benefits and strategic positioning. Yet cloud adoption would create potential or immediate obsolescence scenarios for currently retained organisational IT assets. The scope of retained assets is wide-ranging, but may include quantifiable facets of hardware, software and network components. Dependant on organisational size dynamics, loss of value caused by technical debt and obsolescence may be decidedly unattractive. To delay the burden of technical debt, the reconfiguration or repurposing of assets is recommended. The platforms found to be at risk for technical debt accretion, are expected to be Microsoft operating system environments and supporting infrastructure. A case was made in the survey literature, to repurpose platforms in the form of scalable grids, which continue to support and align with the current organisational need.

4.1.1.1 Conceptual Design Assumptions

The repurposing of currently retained IT assets in the form of on-demand grids would require assumptions about the conceptual effort involved. The potential benefit proposed by a grid alternative, versus the cost of the technical debt manifestation, would logically be of concern to decision makers. The flexibility and utility of the design would further require articulation of expectations in respect to the constituent grid components. To increase the potential worth and appeal of a conceptual design solution, the formation of the grid is suggested to approximate a zero investment. A plausibly optimum design solution, should repurpose existing hardware, software, network and supporting infrastructure, with little or no, administrative or procurement overhead. The primary commitment is to reuse legacy and current operating systems, computational platforms, functional software and transmission devices. The envisioned grid design is to be autonomous and not predisposed to current deployment topology, security framework, computational architecture or administrative configuration. By implicit design, the planned grid prototype ought to be non-persistent and truly dynamic.

4.1.2 Treatment Design

The design artefact should ultimately facilitate HPC and HTC hybridized functionality within a singular construct. During the literature investigation, it was uncovered that design obstacles of inherent parallelism, abstraction, restrictive practice and heterogeneity are evident within the currently obtainable solutions. Hence, the feasibility of an innovative design solution requisites cognisance of design requirements that have bearing on these obstacles.

Drawing from the literature survey, the consensus was that parallelism should be sought at every available opportunity within an HPC and HTC design solution. Parallelism brings about maximization of function and efficiency, begotten from the underlying computational platform. Importantly, efficiency and efficacy correlations are evident around the theme of parallelism. The first step in creating mechanisms for parallel operations is through detailed discovery of the execution environment. In addition, literature informs that abstraction adds deployment complexity and execution overhead, whilst simplifying user interactions and enabling heterogeneous computing.

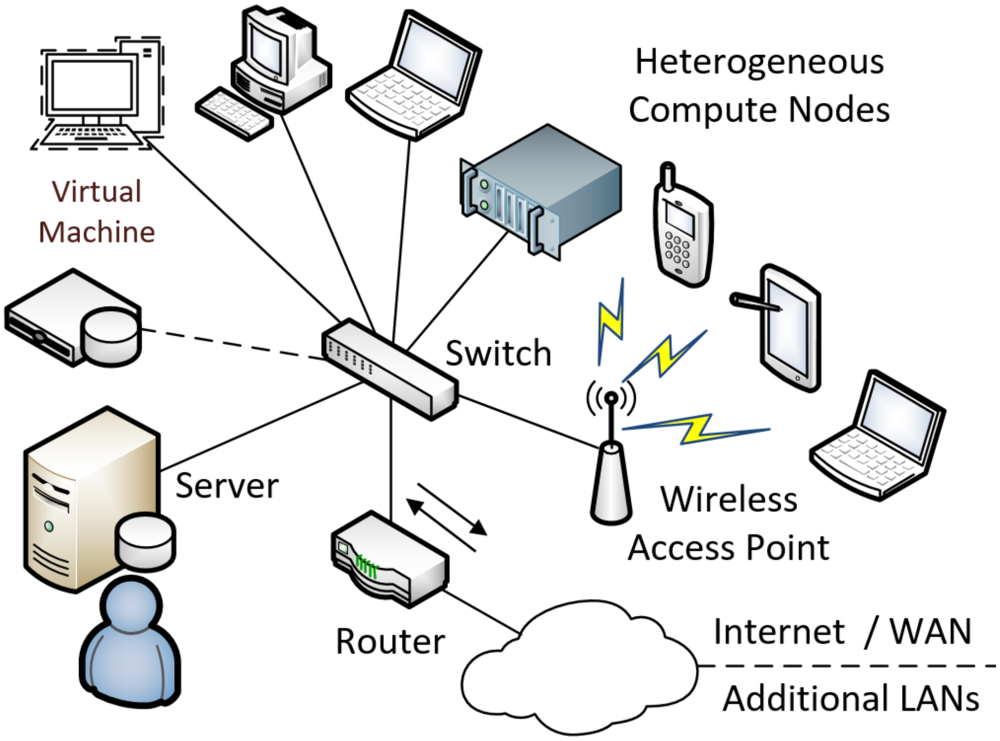


Figure 18 Conceptual design of a dynamic grid

On the other hand, when considering conflicting goals of efficiency versus convenience, the evidence points to efficiency in execution to be paramount within HPC and HTC. A suitable design solution deductively requires exposing diminutive levels of abstraction.

The HPC and HTC heterogeneity problem seems not to have enjoyed holistic coverage in the survey literature. Then also, the available open source and commercial solutions later examined conclude as having perceived homogenous characteristics. The restrictive design practices of currently available solutions, which limit compatibility and choice, could even arguably be self-inflicted. The motivation for placing restrictions via inherent design might have speculative connotations of specialization or financial incentive. In answer to the design for heterogeneity issue, the approach taken in this research is to find common denominators of platform aggregates. The dictum being that commonality indicates points of compatibility without additional restriction. As potential point of departure, consider that Windows operating systems are by enlarge backward compatible, irrespective of underlying processor architecture. Pursuing the lowermost kernel supported API of the operating system aggregate and architecture feasibly produces compatibility across heterogeneous Windows platforms. A supplementary advantage of this approach is that the application technology stack is specifically less abstracted, as revealed in Figure 19.

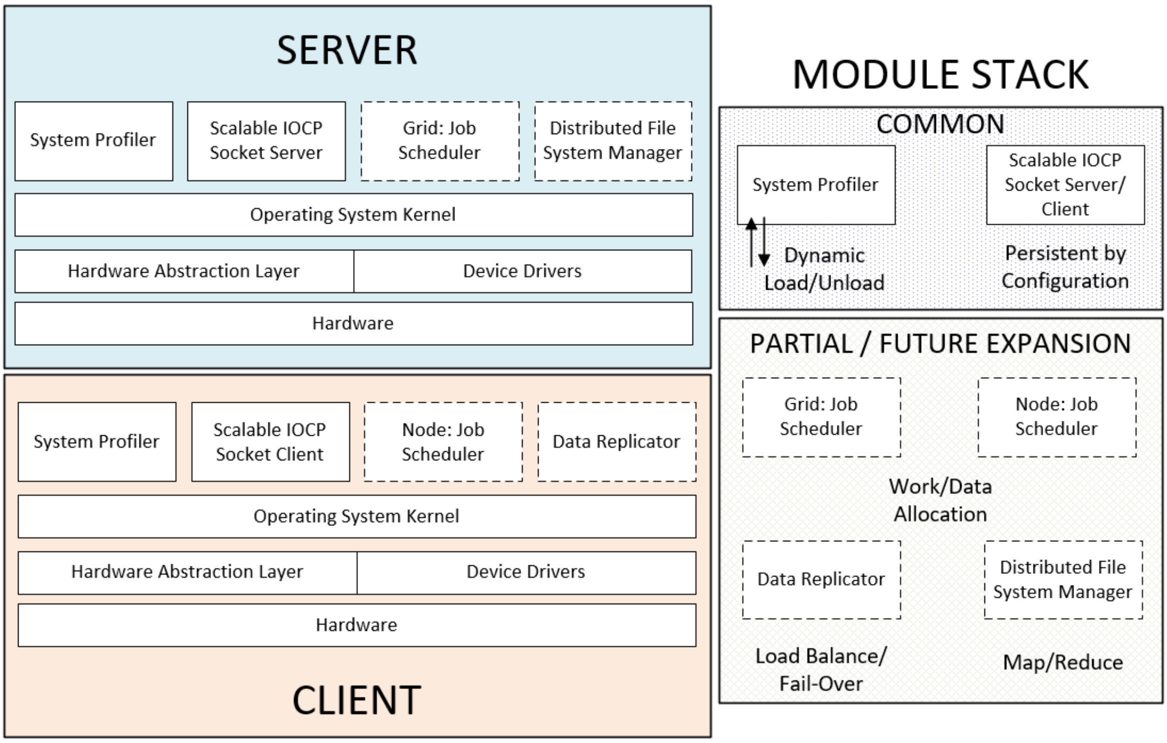


Figure 19 Design technology stack

The proposed design solution is depicted as modular software. A server grid node accepts connection requests from computational client nodes and integrates the collective to form the grid. Each node within the grid is initially profiled to ascertain underlying computational potential, which is reported to the server. The server node in turn, calculates the amassed grid potential. The HPC and HTC work-products or jobs, are submitted to the grid for computation in the form of targeted binary executables. The design choice of including targeted binary executable support, provides for several key opportunities. The primary reasoning is that the restrictive programming environmental constraints of present HPC and HTC solutions may feasibly be overcome. Using existing skillsets and tools, current software assets that provide for data analytics, can viably be reused or reconfigured, to more efficient executable forms. Principally, an executable binary allows for external performance profiling. Moreover, curtailed abstraction may be facilitated by eliminating third party dependency. Likewise, the need for parallel programming skillsets is characteristically negated, as computational node targeting and management may yield optimum parallelization utility. The intent is to support singular or multiple functional binary executables, each matched intimately with the computational platform designated for execution. A client node’s job scheduler would via configuration options, generate hardware thread attuned execution environments per binary workload.

The modular design solution framework furthermore, should allow for the creation of the hybridized data management function. The data manipulation efficiency requirements of HPC and HTC, mandates data sourcing that improves locality of access. For HPC the data sourcing needs are normatively node-local, whereas HTC could additionally exhibit data locality needs as external or near-local. In the context of Windows platform environments however, the barriers to implementation of data locality features becomes problematic. The support compatibility inherent to the Windows OS, is notably due to version and edition. The design arrangements for known solutions to the data locality problem such as DFS, would therefore be determined by the constituent Windows OS platform interactions. Importantly, the DFS participants and functionality in an arrangement, could create implementation difficulties for a potential design solution. The constraints imposed are in reality not just particular to Windows OS versions and editions, but normally encompass administrative and security group membership as well. It would therefore be unreasonable to design for existing DFS environmental support, within the identified technical debt and heterogeneous context. The proposed design solution would require design elements that resolve data locality and scheduling issues, irrespective of the Windows OS runtime environment.

4.1.3 Treatment validation

The statistical data for global desktop operating system sales of the last decade, revealed that the Microsoft Windows operating system platform was at most probable risk for technical debt formation. In order to delay a technical debt burden, the reconfiguration or repurposing of retained IT assets is recommended in literature. The platform constituents of the Windows infrastructure, which may expose technical debt or eventual obsolescence attributes, are consequently of interest in formulating a repurposing strategy. In planning for IT asset redeployment, investigations are mandated in order to identify specific enabling opportunities that reduce or delay technical debt. The exploration of the Windows platform components of hardware, software and network infrastructure are by virtue encompassed.

The physical hardware environment supported by the Microsoft Windows OS, incorporates proprietary patterns for server, virtual machine, workstation or desktop installations. Of prominence however is that any installation prerequisites CPU instruction set compatibility with the Intel® (henceforth Intel) *x86* or *x64* architecture. The x86 architecture family, describes an arbitrary CPU platform that has backward compatibility with the Intel 16-bit and/or 32-bit instruction set. Equally, the x64 architecture family signifies backward compatibility with Intel's 64-bit instruction set. The backward compatibility of the x64 architecture is notably extended to also include x86 architectures. The specific backward compatibility of a given platform, is not however guaranteed for any particular qualifying CPU architecture.

The difficulties in achieving backward compatibility within CPU architectures are importantly due to the heterogeneous nature of CPU brand and supporting instruction sets. Consider the addition of CPU evolutionary features, which inherently necessitates the brand manufacturer to amend the underlying instruction set. Should a hypothetical software product utilize the newer CPU features, the essential architecture support and compatibility thereof would realistically become fixed. Counter to backward compatibility, the software could possibly only achieve future CPU instruction set support. Of significance to a potential design solution is that the potential future technical debt hardware and software environs, could credibly be supporting varied x86 and x64 architectures. The motivation for abstraction as answer to the heterogeneous platform problem, is known to be well served by degrees of complexity inherent to the scope of the development effort. The addition of abstraction to the proposed design, is however questionably regarded as indolent and efficiency defeating.

With the aim of repurposing the holistic technical debt environment, the projected design solution would make use of a minimal base x86 architecture. The commonality of inherent CPU platform instructions, can importantly be considered as solution critical in avoiding abstraction whilst reducing processing overhead. A carefully deliberated x86 architecture design should theoretically derive cross platform and architecture compatibility, for either legacy or modern hardware and software. The proposed design's OS cross platform compatibility is likewise inferred.

The varying Microsoft Windows OS editions and versions are for the most part backward compatible. The customary method of gaining cross platform and architecture software compatibility for any arbitrary OS, advantages commonality of exposed kernel API. The differing kernel API compatibility and feature support facilitated between editions or versions of the Windows OS, are indeed readily overcome for trivial software development scenarios. The problem dimensions however increase dramatically for multifaceted software endeavours. As discovered in the literature survey, enforcing restrictive practice definitely also has a role to play. Even so, the use of abstraction would add undesirable processing and throughput overhead, which detracts from the efficacy and efficiency sought. The projected design solution would therefore make use of minimal base API compatibility, to achieve Windows OS cross platform, network and architecture support. Pursuing the lowermost common OS kernel API aggregate, should viably safeguard compatibility whilst maximising performance and utility across the entire technology stack. The proposed design's transformation effort would however significantly increase during the initial development stages.

The design approach that seeks architecture instruction set and kernel API compatibility, as enabler of holistic cross platform integration is surely not new. However within a technical debt context, the rational of the approach is plausibly weighted due to the potential benefit return. Besides platform integration, flexibility in functional deployment and predictable run time performance may be realized. This design approach per illustration, permits the use of fat executables. A shrewd application design may have multiple procedural calls that for fill the same function but implement differing instruction sets. The application could then dynamically exploit the aggregate instruction set to uncover features and efficiencies. Knowledge of the underlying architecture and OS would nevertheless be key.

4. 2 The Execution Environment Context

4.2.1 Problem Investigation

The organisational need is best served by efficiency and effectiveness of the utility. The utility of a HPC and HTC grid, would seek idealized use and management of potentially diverse organisational processing resources. However the actuality of diverse resources, under normal conditions presents significant difficulties in application. The heterogeneity problem is meaningfully persistent. Certainly, information regarding the environmental execution context can play a key role in achieving higher levels of effectiveness and efficiency. Yet, enacting environmental discovery and leveraging resources for potential benefit is known to be a non-trivial exercise.

The organisational environmental scope, dauntingly yields differing resource characteristics and attributes at every level of the technology stack. To abridge complexity and allow integration, the identification of environmental characteristics that produce actionable parallelism opportunities is recommended. The environmental contextual features of interest, are the recognized causes of processing and throughput bottle-necking.

4.2.2 Treatment Design

The processing platform characteristics that influence efficiency and efficacy as grid participants, was motivated in the literature review. A suitable design artefact would then logically require the environmental concerns to be addressed. The platform characteristics of foremost concern are: CPU instruction set, CPU topology, cache sizes and associativity, RAM and secondary storage interaction, OS provided features, network configuration and media support.

The currently available open source and commercial system profiling solutions, described in Table 10 and depicted in Figure 20, are problematically not particular to both HPC and HTC deployments. The environmental detection features which are delivered by these solutions, could however add conceptual input design value. The solutions scrutinized, are noted to be sensitive to administrative security environments, frequently owing to the use of ring zero kernel drivers. Undesirably also, these solutions habitually demonstrate lengthy execution run-time dynamics. Some of the investigated system profilers, are observed to have programmable interface features that enforce restrictive prerequisites. But most importantly, the considered system profiling solutions do not wholly address the areas of contextual design as highlighted in the survey literature.

Table 10 System profiling solutions

ATTRIBUTE	OpenMPI hwloc / nwloc v2.0.1 Windows	CPUID™ CPU-Z v1.84.0	REALiX™ HWiNFO V5.74	OpenHardware Monitor v.0.8b	Apache® Hadoop YARN v2.8.0 Node Manager*
RESOURCE PERSPECTIVE					
Functional Execution Wall-clock Time (sec)	0,479	3,998	6,326	2,889	-
Process Memory Working Set (bytes)	5 840 896	12 447 744	52 695 040	44 105 728	-
Functional on Disk Size (bytes)	1 441 792	-	1 133 680	270 336	Java Runtime
GUI on Disk Size (bytes)	2 289 664	3 555 328	4 235 264	1 327 104	Java Runtime
Additional Dependencies	Infiniband Fabric	None	None	.NET Framework version 2.0	Other
Ring Zero Driver	Yes	Yes	Yes	Yes	No
License	Berkeley Software Distribution (Clause 3)	Freeware & Commercial End-User License Agreement	Freeware & Commercial End-User License Agreement	Mozilla Public License V2.0	Apache License V2.0

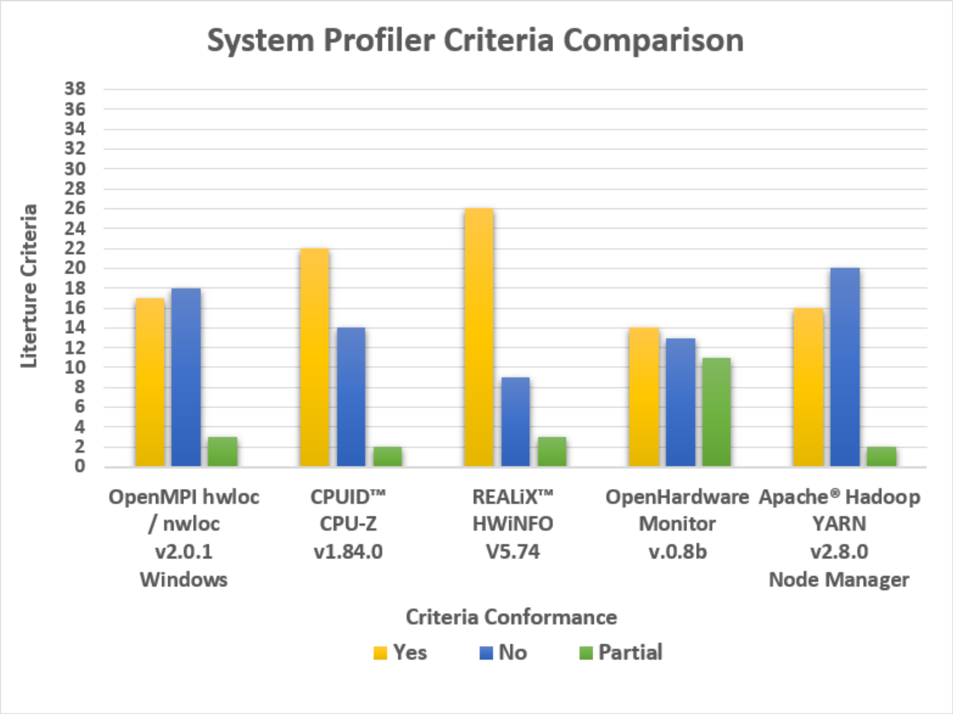


Figure 20 System profiling criteria comparison

A detailed comparison of the literature review motivated HPC and HTC profiling criteria, has been presented in Appendix A for perusal. Of likely interest, the review authors Broquedis, et al. (2010) are the original creators of OpenMPI's hwloc system profiler.

The identification of supported CPU instruction set can provide exploitable avenues for parallelism. The instruction sets that support SIMD and MIMD, are reiterated to be especially desirable in achieving effective and efficient parallel operations. The identification of the instruction set, further enables architecture enumeration, which can be important in gaining software compatibility. The detection of the CPU topology, principally relates logical processing units and hardware threads, which can be intimately controlled for utility. The topology attributes of the CPU, are correspondingly able to qualify and schedule performance workloads in a useful manner. Likewise, understanding of the caching hierarchy of the potential compute node, is known to be critical in planning instruction and data allocation across the CPU addressable processing units. A point of departure for an operational cache management strategy at compute node level, may thus be formulated.

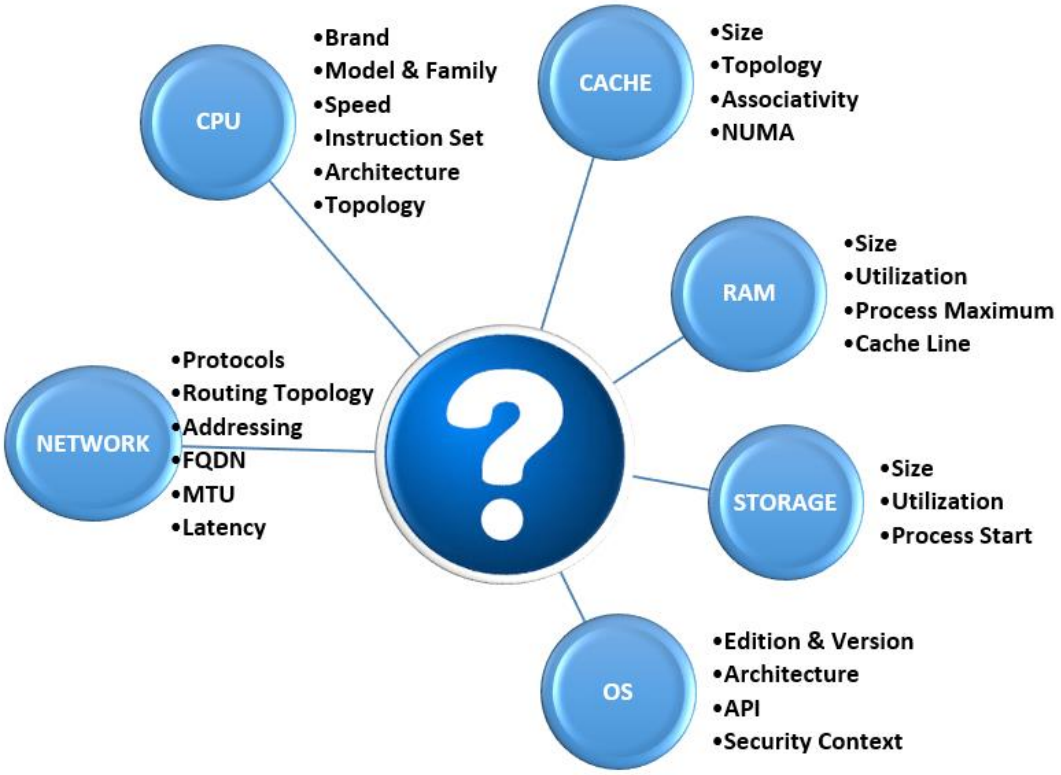


Figure 21 Environmental characteristics of design

The close relationship between RAM and secondary storage, allows programmable assessment of the system state. The detection of current RAM utilization and free storage in itself was found to provide an incomplete overall picture of the real-time system state. Nevertheless, such information is potentially adequate, in an initial workload placement decision.

Identification of the OS discloses edition and version information. The API programmability, features and inherent underlying OS hardware support, affords a contribution towards establishing integration and software compatibility. The detection of the OS network API suite per example, may determine if zero copy features and egress offload are supported. Finally, the enumeration of physical and logical network environmental configurations, allows for planning of dynamic global and node-local data placement, in addition to enhanced bandwidth consumption.

4.2.3 Treatment validation

To form a HPC and HTC grid, the design artefact should gather information about the environmental architecture, OS and network capabilities. A fuller list of requirements together with feasible motivations, are available in Appendix B for assessment. Within the design purview, the requisites for information gathering could be understood to have static and dynamic elements. The information sourcing needs of the anticipated design solution are importantly mutable. The system profiling solutions examined, although not specifically built for hybridized HPC and HTC, do however provide relevant insights to the problem.

The observed profiling solutions are repeated to use mechanisms of API interfaces and in most instances ring zero drivers. Under normal circumstances any third party API's programmatic communications, requires the in-memory instantiation of its function. The same could be said of ring zero drivers that enable amongst others, the use of kernel privileged features. However the design choice of a ring zero driver additionally entails deployment complexity, development and administrative commitments in addressing the solution's security environment.

In terms of physical instantiation, an exposing API design solution is found to be problematic for the environmental discovery context. The ratio of static versus transient platform information required, is noted to be significantly skewed towards the static for both HPC and HTC utility. The mutable information need for near real-time workload placement decisions, is considered to be diminutive when compared with the static information requirements. Meaningfully, the bulk of the information need is geared towards holistic platform profiling, which enables targeted workload scheduling. Much of resources occupied by a profiling solution, would therefore become immediately wasteful after the workload scheduling requirements were met. The in-memory and dynamic processing load of a singular API design solution, cannot then be easily justified.

The proposed design solution accommodates a system profiling module that dynamically loads and sheds its function, allowing for resource reclamation. Only the all-inclusive result of static system profiling is to be captured within a minor data structure and made available for future enquiries.

The anticipated design’s real-time system profiling need, is met using a separate thin profiler, embedded within the compute node’s job scheduler. Intentionally the marginal and mutable information need will be locally serviced, where its resource processing and storage overhead can be better justified.

The projected design solution further recommends not making use of ring zero drivers and kernel privileged instructions. The stated norms of the design problem context are reminded to simply disqualify the addition of deployment complexity, security and administrative overhead. The functional expediency of ring zero drivers would therefore involve substitution, by supplementary development effort to gain comparable solution outcomes. The feasibility of this specific substitution, regards making innovative use of the backward compatible Windows API and architecture instruction set functions.

Another issue explicit to the popular HPC and HTC profilers reviewed, is that of perceived information quality. A study of both these system profilers, reveals that the platform environmental data is principally sourced from the presiding OS. The design intent of relying on the OS as primary enumeration source, is then also prominently motivated as ensuring cross platform heterogeneous support. Only within the HPC profiler can a CPU instruction set backend, supplement the activity should OS support be found absent. However, on further analysis conspicuous limitations of the employed design choice in these solutions become apparent. By implication, the processing sequences that gather environmental information in both solutions, are exclusive and produce singular answers per instantiation.

Consider that the OS may well inaccurately abstract or interpret the hardware layer. Undoubtedly virtual machine, improperly configured hardware or OS environments, could easily yield such erroneous data on runtime instantiation. The OS provided environmental information is importantly not necessarily reflective of the underlying platform reality. The environmental data sourced by means of an instruction set backend, is for the same reasons considered to be unreliable. The detected hardware reality, significantly cannot by itself produce an accurate account of the supported OS runtime environment.

The proposed design alternative, makes use of both instruction set backend and OS environmental sourcing, often sampling the same information for subsequent reporting or comparison. A more accurate picture of the run-time environment as a result, may feasibly lead to better processing decisions.

Discernments can be made around the generalizability of the proposed design's system profiling function. When disregarding substantial portions of the research context, the potential design remains useful in the accurate determination of generic platform environments. The scope of information afforded by the proposed design, can convincingly be applied as remedy to wide-ranging problem scenarios. Per brief illustration, consider circumstances such as: asset inventorying, software license enforcement, holistic topology discovery, environmental problem detection and isolation.

Additionally, the proposed design's profiling features are attractively resource savvy. By separating the static and mutable information needs, the processing and storage requirements of the design solution takes on minimalistic dimensions. Moreover, avoiding internal design adoption of kernel ring zero and API functionality, makes the solution non-specific and potentially extensible to other OS distributions.

4. 3 The Network Utility Context

4.3.1 Problem Investigation

The control of organisational IT assets concerns the functional administrative deployment and configuration, of the network environment by which utility can be acquired. How to generate capacity for data analytical utility and at what cost, would be of interest to organisations. A potential data analytical solution should logically be investigated to establish its value proposition. The detailing of the endeavour cost and benefit extents, would reveal the realistic effort and total cost of ownership metrics, used as criteria in the acquisition decision.

The configuration and deployment of a contemporary HPC or HTC solution, requires varied and often complex processes to be effected. The design homogeneity within a multitude of available solutions, could conceivably confine future choice after the initial purchase. Once implemented, the solution may have observably static characteristics, requiring additional effort and expense to conform to shifting organisational need. An exacerbating factor to consider includes the fact that commodity support claimed by solutions, are predominantly not meant to imply inexpensive infrastructure would be supported. The total administrative control over the solution's execution environment is often then also mandatory. Conceivably a deployed solution not actively instanced, would integrally continue to occupy potentially useful resources. The actual expense and effort incurred by an HPC or HTC solution may well be difficult to isolate. The projected solution's network context is for these reasons a highly relevant design topic.

Based on derivative stakeholder requirements, an ideal design solution would have low overall fiscal and administrative cost, whilst still preserving organisational flexibility. The solution would continuously promote support for diverse infrastructure, yet not passively consume physical resources. A potential design solution, necessitates counsel on how network deployment features may meet these requirements and still maintain utility and function.

4.3.2 Treatment Design

On inspection of existing HTC and HPC design solutions, described in Table 11, the seeming fulfilment of the raw composite requirements of design is not reflected. The problem context is the most credible underlying influence in failing to meet these requirements. The distinctive nature of contemporary design solutions, is objectively inert and unwieldly.

Table 11 Contemporary HPC and HTC solutions

FEATURE	OpenMPI V3.0.0	Apache® Hadoop v3.1.0	Microsoft® HPC Pack 2016 Update 1 5.1.6086.0	UW Madison HTCondor v8.6.10
CORE SOLUTION PERSPECTIVE				
Type	HPC Library	HTC Platform (Build from Source)	HPC Platform & Library (MS-MPI)	HTC Platform & Library
Pre-requisites	-	- Java SDK / Runtime1.6 - Maven 3.0 - Windows SDK - CMake 2.6 - GnuWin32	Head Node - Win Server 2012 R2 Compute Node - Win Server 2008 R2 SP1 Workstation Node - Win 7 (x86 & x64) Cluster Database - SQL Server 2008 R2 .NET Framework 4.6.1	- Visual C++ 2012 Runtime
Minimum System Requirements	-	- Win 7 - x64 Architecture	- x64 Architecture (Roles) - 4 CPU Cores - 4 to 8GB RAM - 50 GB HDD	- Win Vista - x86 & x64 Architecture > 300 MB HDD
Native Programming Language Support	C, FORTRAN & C++	Java, Python & C++	C, FORTRAN & C++	Python & C++
Unpacked Size on Disk Prior to Installation	106 MB +	168 MB +	1 063 MB +	124 MB +

A proposed design solution recognizes that socket derivative messaging is at the heart of IPv4 and IPv6 network communications. The modern manifestations of HPC and HTC solutions, take advantage of various forms of SAN socket provisioning, backed by dedicated network infrastructure. An appropriately configured SAN or RDMA enabled platform, can notably achieve higher HPC or HTC efficiencies and throughput. Ordinarily when using SAN and related technologies, as attested to in the literature survey, the execution environment is transparent to a functional application.

However, within the problem context, an arbitrary organisational network and supporting platform reality would be indistinct. Opportunely SAN or RDMA technology support is inferred not to be critical to a design solution. The main features of existing SAN and RDMA enabled solutions do however provide problem context awareness for a potential design. A rational design pattern should impose obligation to seek performance-enhancing features exposed by the network infrastructure and execution environment.

The aforementioned stakeholder network context requirements, could possibly be met with an IOCP design treatment. Primarily the implementation mechanisms of IOCP socket operations are programmatically compatible with the majority of commodity operating systems and supporting infrastructure. During instantiation, the design artefact is suggested to completely enclose the IOCP functionality as a software module. The anticipated design outcome is that artefact instantiation will not impede deployment flexibility, nor consume passive resources outside of grid formation. Furthermore the design provisioning of overlapped IOCP socket handling, is reported in literature to significantly reduce overall processing overhead. An overlapped IOCP socket scheme would intrinsically progress in a non-blocking fashion. The application processing overhead attributed to network socket communications, could thus be considerably reduced. A proposed design that includes overlapped IOCP socket services, presents supplementary opportunity. When mapped onto hardware threads, overlapped IOCP is known to facilitate highly scalable and available network environments.

The potential design solution, as shown in Figure 22, would couple system profiling and run-time IOCP services configurations. Such a design should allow for the dynamic shaping of the communications environment whilst complementing the physical network reality. The design may well leverage the run-time elasticity of the network environment to achieve higher performance and throughput. Per illustration, knowledge of the network MTU combined with an attuned memory transfer configuration, could facilitate beneficial use of larger transmission frames.

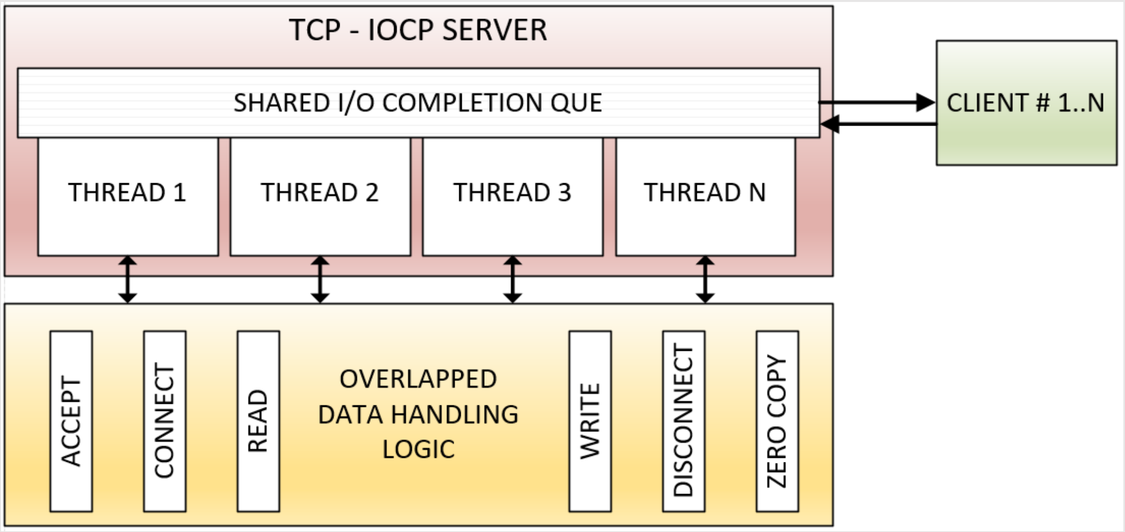


Figure 22 Design of overlapped I/O completion port model

4.3.3 Treatment validation

The prevalent HPC and HTC solutions available, routinely delineate implementations of mainly two inter-process communication schemes. The discernment of inter-process communications describing locally hosted or remote application processes that require interaction. HPC enactments classically adopt the Message Passing Interface (MPI), whereas the HTC solutions gravitate towards the use of Remote Procedure Call (RPC). Of importance would be that the functional mechanisms of the two schemes, have direct relationship with the characteristics of the utility supported. The understandings produced by probing the underlying mechanisms of MPI and RPC, potentially could be crucial information to amalgamate HPC and HTC onto a single platform.

The MPI communication scheme, concerns application development and execution scenarios that use parallel computing architectures as platform. An enabled MPI application, conspicuously features a distributed memory or distributed shared memory model. In brief, the application's processes are designated for execution onto a number of logical processors within a larger virtual topology. Each application process having access to its own private memory store (i.e. distributed memory). The application process could also share a logically addressable memory store with another related process (i.e. distributed shared memory).

The MPI developed application would remain reliant on the specific build implementation for execution. A normative MPI distribution, unpacks as components, frameworks and modules that expose APIs. Modest arrangements are fundamentally portable tools and core programmable libraries. The goal of the MPI library is to facilitate the abstraction of an application's computational environment, inter-processor, network socket and supporting protocol services. When used in conjunction with a supported programming language, the MPI library would allow a software utility to transfer messages between spawned constituent application processes. Further prominent traits of MPI, include ad-hoc atomic thread safe data interchange and process synchronization.

The RPC communication scheme, concerns application development and execution scenarios that use distributed client-server architectures as platform. An enabled RPC application allows for the invocation of software subroutines regardless of actual code and data locality. Similar to the MPI scheme, RPC is a programming language level construct that requires compilation or run-time interpreter support. The transparency provided by RPC, is fundamentally based on the ability to uniquely address the procedural space within a supported application.

The RPC application communicates with its constituent distributed components, by marshalling parameters and procedural handles into serialized forms. Dependant on the address space of the invoked application subroutine, the RPC mechanics encapsulates and reroutes the request to the unique addressee for un-marshalling. The destination can importantly be either an abstracted socket derived network node or locally hosted process. The RPC memory model is typically more flexible, yet could requires more storage and processing overhead than that of MPI. By default, each RPC request allocates and deallocates shared process memory on a node by node basis. Other shared memory configurations could include combinations of node confined persistent and dynamic contiguous allocations. Thread safety for RPC is achieved through explicit binding of process handles and serialization of the resultant execution requests.

The points of commonality for MPI and RPC, are derived to necessitate programming language level support and the ability to uniquely address the constituent application processes. Likewise, both schemes abstract the network socket environment to facilitate remote process communications. The contrasting themes of interest within both schemes, involves perspectives on memory locality and thread safety. The MPI scheme encourages memory locality and shared data integrity at logical processor level. The RPC scheme opposes, by sharing memory across dissimilar locations and implementing thread safety through serialization.

The actual workload capabilities envisioned for the design platform, is reaffirmed to ultimately cater for specialized HPC, HTC or hybridized composites. Consider then the application environment of the two dissimilar inter-process communication schemes of MPI and RPC, whilst also bearing in mind the restrictive requisites the use of such would enact. The salient concerns are of prescriptive programming languages, libraries, OS, runtime environment and supported network fabric. An idealistic design solution within the research context, should seek to reduce restriction within the execution environment, by maximizing the flexibility of choice surrounding the utility.

The proposed design as offered, uses the IOCP overlapped apparatus to establish grid formation. The server vehicle of the design, can feasibly enable efficient command and control of the grid platform’s resources. The client portion of the design, responsible for the targeted processing of workloads. An IOCP design implementation would purely perform insular OS level integration of the server and compute clients. The design’s intent is not to dictate how an application workload would achieve its utility. This approach could viably increase the tractability of classifiable workloads supported.

By not dictating the inter-process and memory mechanism of a workload utility, a number of potential support scenarios may evolve. As the platform and the workload application are on the whole insular constructs by design, the following utility choices become available:

- The programming language for development, would only be constrained by the ability to compile an instantiating stub executable
- No pre-emptive restriction are placed on the infrastructure, security or administrative environment
- Prerequisite utility configuration and divergent memory models become possible
- Implementing thread safety could make use of any relevant mechanism
- Single threaded workloads that do not share memory, may take advantage of full computational node parallelism

The shared characteristics of MPI and RPC, to uniquely address application processes, could also feasibly be duplicated within the proposed design. A server instance would dynamically configure and publish a data structure for consumption by compute nodes through use of IOCP sockets. The structure containing detailed information regarding the timed individual application process' state and locality. A client node would declare the data structure locally, for access by the application utility processes. The method of data structure exposure, being the universally supported OS API of non-persisted memory mapped file. The utility's inter-process and memory sharing mechanics could then use the information to facilitate global inter-process communications.

4.4 Chapter Summary

Chapter 4, discussed the artefact design endeavour in terms of problem contexts. Within the research descriptive, a problem context was invariably referenced and related back to the survey literature. Of importance was that each problem context recognized, conceptually informed rational responses to the initial research questions posed. The primary contexts of technical debt, execution and network environment, deduced stakeholder requirements and concerns within the larger research problem domain. The requirements of each isolated context, formulated selection and defence of solution design choices. The problem context and related requirements, made design comparisons with existing technologies, schemes and solutions. The discussions surrounding the validity and generalizability of design, were intended to serve as measure in assessing the overall research contribution.

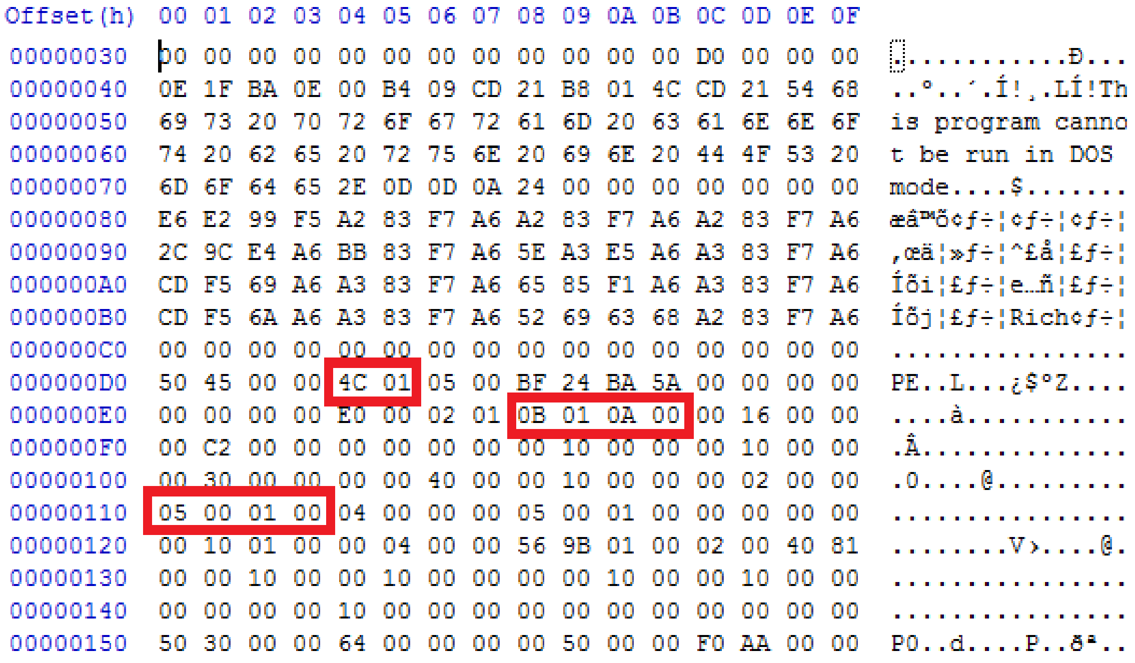
CHAPTER FIVE
FINDINGS, DISCUSSIONS & LIMITATIONS

5.1 The Technical Debt Context

The research investigations conducted, imparted that the Microsoft OS supported platforms would be at highest potential risk for future technical debt manifestations. The observed statistical data in Figure 4, moreover relating the scope of the OS versions from which technical debt can conceivably originate. Meaningfully, the market share data showed established support for OS platforms, ranging from versions and editions of Windows XP up till Windows 10. The supported instruction set architecture of a Windows platform OS, was also reputed to be unclear within a realistic technical debt environment. The available HPC and HTC solutions gaged within the research descriptive, were debatably never designed to function within truly heterogeneous technical debt environments. In answer to partial heterogeneity, the observed HPC and HTC solutions relied unanimously on abstraction schemes and/or restrictive practices.

5.1.1 Heterogeneous Operating Systems

The prior research design treatment and validation discussions, motivated OS API backward compatibility to integrate heterogeneous platforms. Plausibly the lowest backward compatible OS kernel release supported, could establish cross platform integration. The design artefact establishes platform support starting from the Windows NT 5.1 OS kernel (i.e. Win XP build 2600 & Win Server 2003 build 3790), which notably encompasses all the Windows platforms identified within the research data. The significant implications of the design's kernel provisioning are that extended compatibility is achieved across subsequent server or desktop versions of the Windows platform suite. Importantly, architecture compatibility is accomplished by exploiting the x86 32-bit architecture, which promotes basic instruction set operability with editions of the Windows 64-bit platforms. Predictably due to the solution compatibility choices deliberated and implemented during development, the artefact theoretically achieved compatibility with Linux POSIX compliant platforms as well. A Linux system installed with the WINE (Wine Is Not an Emulator) open source package, should natively allow the design artefact's execution. The following figure illustrates the prototype artefact's initial portable executable header, as viewed in a typical hex file editor. Recognizably, as evidenced in Figure 23, the artefact compilation is meant for execution on a 32-bit Intel compatible architecture machine and x86 Windows platform (i.e. values 014C, 010B & 0A00 hex). The portable executable header declaring the OS kernel attributes, conforming to compatibility with Windows NT 5.1 (i.e. values 0005 & 0001 hex).



```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000030  00 00 00 00 00 00 00 00 00 00 00 00 D0 00 00 00  .............Ð...
00000040  0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68  ..º..´.Í!¸.LÍ!Th
00000050  69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F  is program canno
00000060  74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20  t be run in DOS 
00000070  6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00  mode....$.......
00000080  E6 E2 99 F5 A2 83 F7 A6 A2 83 F7 A6 A2 83 F7 A6  æâ™õ¢ƒ÷¦¢ƒ÷¦¢ƒ÷¦
00000090  2C 9C E4 A6 BB 83 F7 A6 5E A3 E5 A6 A3 83 F7 A6  ,œä¦»ƒ÷¦^£å¦£ƒ÷¦
000000A0  CD F5 69 A6 A3 83 F7 A6 65 85 F1 A6 A3 83 F7 A6  Íõi¦£ƒ÷¦e…ñ¦£ƒ÷¦
000000B0  CD F5 6A A6 A3 83 F7 A6 52 69 63 68 A2 83 F7 A6  Íõj¦£ƒ÷¦Rich¢ƒ÷¦
000000C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000D0  50 45 00 00 4C 01 05 00 BF 24 BA 5A 00 00 00 00  PE..L...¿$ºZ....
000000E0  00 00 00 00 E0 00 02 01 0B 01 0A 00 00 16 00 00  ....à...........
000000F0  00 C2 00 00 00 00 00 00 00 10 00 00 00 10 00 00  .Â..............
00000100  00 30 00 00 00 00 40 00 00 10 00 00 00 02 00 00  .0....@.........
00000110  05 00 01 00 04 00 00 00 05 00 01 00 00 00 00 00  ................
00000120  00 10 01 00 00 04 00 00 56 9B 01 00 02 00 40 81  ........V›....@.
00000130  00 00 10 00 00 10 00 00 00 00 10 00 00 10 00 00  ................
00000140  00 00 00 00 10 00 00 00 00 00 00 00 00 00 00 00  ................
00000150  50 30 00 00 64 00 00 00 00 50 00 00 F0 AA 00 00  P0..d....P..ðª..
```

Figure 23 Artefact portable executable header

The solution's facility to realize cross platform and architecture compatibility, is asserted to meet with the heterogeneous platform requirement of a technical debt context.

5.1.2 Security Context and Abstraction

The generic computational environment is classically managed by instruments that segregate the rights and privileges assigned to authorized user groups, resource objects or individual members. The normative assignment of rights and privileges, pertaining to the minimal assignment possible that still allows the intended functional role to be accomplished. The administrative role, stereotypically reserving the rights to arbitrate platform configuration changes. A recurring security contextual problem, which impacts the proposed solution may as a consequence be identified. The act of software deployment and installation is customarily regarded as a platform change, hence necessitating administrative arbitration and overhead.

The deployment of contemporary HPC and HTC solutions surveyed, is reminded to exhibit fundamental reliance on pre-requisite administrative oversight. The use of technology stack abstraction, is conversely central and generic to modern software development practice. The examined manifestations of HPC and HTC, potentially escalating degrees of prior discernment regarding abstraction. To attain function, the studied HPC and HTC build environments principally added or operated on multiple layers of abstraction.

A proverbial paradox is engendered, in that efficacy and efficiency within an HPC or HTC solution, are deemed major characteristics of the utility. Yet owing to internal design abstraction, opportunities for improved performance and proficiency of resource utilization are being reduced.

```
.686                    ; Intel Pentium Pro compatible instruction set, 32-bit
                        ; - non-privileged instruction set mode
.model flat, stdcall    ; 32 bit memory model, standard call convention
option casemap :none    ; case sensitive
. . .
```

Codelet 1 Artefact environmental declarations

The prototype artefact registers an observably minimalistic technology stack. An artefact instantiation, achieving autonomous and encapsulated run-time directly above the OS layer. Prominently the solution meets the reduced abstraction need within a technical debt context. By implication, no third-party libraries, language interpreters or frameworks are required to perform execution. Suggestively neither is administrative intervention required to perform deployment, installation or instantiation.

The autonomous portable executable form of the solution artefact, suitably entails distribution and installation by means of file copying. The assigned rights and privileges of the instantiating user security context, realistically the only inhibitor to utility.

5.2 The Execution Environment Context

The design solution would be required to manage and control, diverse organisational processing resources in order to facilitate grid formation. To gain potential efficiency and efficacy from participant grid resources, the discovery of underlying hardware and software environs was purported as essential. Information regarding resource attributes that promote higher throughput, parallelism and reduced latency computation are of documented interest.

The scope of information about the preliminary environmental hardware characteristics pursued, includes platform detection of attributes for CPU, cache, RAM, secondary storage and network configurations. The range of software attributes, reflecting on OS capabilities and API compatibility. In order to avoid dysfunction, the attributes influencing the solution’s strategy or operational decision making, was recommended to necessitate corroboration from more than one source.

The prototype artefact for both client and server utility, uses a system profiling module that functions as a dynamic link library. The profiler can subsequently be loaded, executed and unloaded on an ad-hoc basis. However the planned instantiation is envisioned to be singular per grid participant node. The environmental context needs of the overall solution, was previously motivated to have static and mutable elements. The system profiling module explicitly addressing the static need.

```
; Detect CPUID Opcode
pushfd
pop     eax
xor     eax, 200000h
push    eax
popfd
pushfd
pop     edx
xor     edx, eax
bt      edx, 21
jnc     @@A1   ; CPU486 > ?
. . .
@@A1:
lea     edi,CPU.Manufacturer
cpuid
mov     CPU.InputValue, eax
. . .
```

```
; Detect CPU Extended
; -Function Support
mov     eax, 80000000h
cpuid
. . .
mov     eax, 80000001h
cupid
. . .
bt      edx, 27 ;RDTSCP
jnc     @@B1
. . .
bt      edx, 29 ; x64Arch
jnc     @@B2
. . .
bt      edx, 31 ; 3DNow!
jnc     @@B3
. . .
```

```
; Detect CPU Basic Info
; & Feature Bits (ex.SIMD)
mov     eax, 1
cpuid
bt      edx, 25  ; SSE1
jnc     @@C1
. . .
bt      edx, 26  ; SSE2
jnc     @@C1
. . .
bt      ecx, 0   ; SSE3
jnc     @@C1
. . .
bt      ecx, 19  ; SSE4
jnc     @@C1
. . .
```

Codelet 2 Using the CPUID opcode to reveal functions and features

The platform hardware attributes are gathered by directly interrogating the CPU instruction set. A leading theme during the interactions with the instruction set, concerns the CPUID opcode. CPUID is conspicuously reliant on derived function and feature values, to produce the desired enumeration attributes of interest. Frequently the attributes availed by the CPUID opcode, requires further processing to enable useful logical transformations.

```
; AMD Specific Topology
mov     eax, 80000008h
cpuid
mov     eax, ecx
mov     ebx, 2
shr     eax, 12    ; ApicIdCoreIdSize
and     ecx, 0FFh ; NumCores
and     eax, 0Fh
mov     edx, ecx
.if (eax)              ; ApicIdCoreIdSize >0?
        mov ecx, eax
        nop
        dec ecx
        nop
        shl ebx, cl
.else                  ; ApicIdCoreIdSize = 0
. . .
```

```
; INTEL Specific Topology
.if (CPU.InputValue >= 11)
call    Funtion11_MASKS
.else
call    Function4_MASKS
.endif
. . .
.if (CPU.InputValue >= 11)
        mov     eax, 11
        cpuid               ; x2APIC in edx
.else
        mov     eax, 1
        cpuid
        shr     ebx, 24   ; get bits: 24..31
        mov     edx, ebx ; APIC in edx
.endif
. . .
```

Codelet 3 CPU Topology brand specific detection

The derived hardware information, contributes initial points of corroboration for the CPU brand specific topology enumeration process. The OS reported information could later be related back to their specific hardware profiling counterparts.

```
; Operating System Information
call     Detect_ADMIN
call     Detect_OS_64
call     Detect_OS_CPU
call     Detect_OS_SYSTEM
call     Detect_OS_MEMORY
call     Detect_OS_AFFINITYMASK
call     Detect_OS_DISPLAY
call     Detect_STORAGE
. . .
```

```
; Detect_OS_64 – determine x86 or x64 OS architecture
Invoke GetModuleHandle, reparg("kernel32")
Invoke GetProcAddress,eax,reparg("IsWow64Process")
.if (eax)
         xchg      eax,ebx
         Invoke    GetCurrentProcess
. . .
.if (eax)
         mov CPU.OSx64, TRUE
. . .
```

Codelet 4 Profiling the operating system

Network configuration discovery is found to be initially contingent on minimal Windows OS platform support for WinSock API version 2.2. The major motivation for seeking WinSock API version support, concerns the literature evidenced obligation for leveraging zero copy network features. Supplementary network information gathered, yields environmental attributes for routing, physical or logical addressing, protocol stack, path MTU, LAN isolation and latency. Often the resultant network attributes are congruently captured in programmable and human readable forms.

```
; Network Configuration & State Discovery
call      Detect_WINSOCK2
.if (CPU.WINSOCK2) && (!CPU.ERR_MEM) && (!CPU.ERR_NET)
          call Detect_PROTOCOL          ; Detect Protocol Stack = IPv4 / IPv6 / ICMP / NetBIOS
          call Detect_ROUTING           ; Detect Logical Addressing/ FQDN / Network ID
          call Detect_LANGROUP          ; Detect Gateway MAC / LAN Isolation
          call Detect_MTU               ; Detect Path MTU and LAN Latency (RTT)
.endif
invoke WSACleanup
. . .
```

Codelet 5 Enumerating the network configuration

The system profiler finally populates a diminutive data structure as the overall static result of a potential grid node’s environment. The invoking module receives the data structure and unloads the system profiler to reclaim resources. A sample of the artefact’s system profiler data structure and visualized output is available in Appendix C. The artefact’s profiling module, observably meets with all 38 comparison criteria used in the contemporary solution review process. Of additional significance is that the entire node profiling result could potentially be transmitted as a single network packet within a default MTU Ethernet packet.

5.3 The Network Utility Context

High performance and throughput computational environments, certainly requisite design patterns that pursue performance-enhancing features wherever possible. A decomposition of contemporary HPC and HTC network arrangements, revealed fundamental dependence on techniques that implement socket derivate communications. However the network related performance-enhancing features specifically employed by present HPC and HTC solutions, do not provide adequate fit with realistic technical debt scenarios. The investigated solutions pointedly abstracted network socket communications and became largely contingent on network hardware acceleration to gain performance. The erstwhile artefact design descriptive of the network problem context, then proposes a software IOCP instrument as alternative for enabling socket based communications. An overlapped IOCP que when mapped onto hardware threads, can markedly provide for highly available and scalable concurrent communication platforms. Coupled with knowledge of the underlying computational platform and supporting network conditions, an IOCP solution argument may advantage additional functional flexibility.

The solution artefact equips both the server and client constructs with the same IOCP module. As the client build takes on the responsibility for eventual workload processing, the IOCP configuration is logically geared towards consuming less resources when compared with the server instance. Irrespective of the node role, the IOCP service can dynamically adjust its configuration to benefit the detected platform reality.

```
; IOCP Server Configuration
BUFFER_SIZE                 EQU 4096        ; I/O Buffer Size in Bytes (Default)
TCP_PORT                    EQU 999         ; TCP Port Number
UDP_PORT                    EQU 995         ; UDP Port Number
MAX_THREADS                 EQU 64          ; Maximum Worker Thread Count
MIN_OVERLAPPED              EQU 5           ; Minimum overlapped RECEIVES per Socket
MIN_ACCEPTS                 EQU 5           ; Minimum pre-created ACCEPT Sockets
MAX_ACCEPTS                 EQU 500         ; Maximum ACCEPT Sockets
MAX_RECVS                   EQU 200         ; Maximum overlapped RECVS per Socket
MAX_SENDS                   EQU 200         ; Maximum overlapped SENDS per Socket
BURST_ACCEPTS               EQU 100         ; Burst by ACCEPT Count
. . .
; Adjusting IOCP Service to the Underlying Platform
mov esi, NODEDATA
mov eax, BUFFER_SIZE
mov ebx, [esi].OSPagesize
mov ecx, [esi].OSCores
mov edx, [esi].wsa_IPv4Address
.if (ecx > MAX_THREADS)                     ; Ensure MAX THREADS or less
        mov ecx, MAX_THREADS
.endif
. . .
```

Codelet 6 A highly scalable IOCP service module

The IOCP configurations of potential interest regard accept scaling and I/O buffer tuning. The per socket I/O memory buffers, are adjusted at start-up to reflect the OS RAM page size of the platform. Accept scaling refers to a contrivance that pre-emptively creates and manages a number of accept sockets in anticipation of new connections. Importantly accept scaling reduces the connection latencies and processing overheads, normally associated with on-demand socket creation.

```
; Dynamic Accept Scaling Ex. Ensuring min of 5 accepts, scaling by 100 accepts, to a max of 500
mov eax, Event
mov Limit, 0
.if (eax == [esi].AcceptEvent)
        invoke WSAEnumNetworkEvents,[esi].hSocket,[esi].AcceptEvent,addr NEvent
        .if (eax == SOCKET_ERROR)
                mov SCKTErrorITM, 20
        .else
                and NEvent.lNetworkEvents, FD_ACCEPT
                .if (NEvent.lNetworkEvents == FD_ACCEPT)
                        mov Limit, BURST_ACCEPTS
                .endif
        .endif
.elseif (eax == [esi].RepostAccept)
        invoke InterlockedExchange,addr [esi].RepostCount,0
        mov Limit, eax
        invoke ResetEvent,addr [esi].RepostAccept
.endif
. . .
```

Codelet 7 Dynamic accept scaling

A curious caveat of the Microsoft socket implementation concerns the desired zero copy and egress offload APIs. The kernel interface requires the APIs to be referenced and invoked via their globally unique identifiers (GUID) at run-time. In the absence of API referencing, the Winsock library is loaded and unloaded per program invocation of these functions. An implementation oversight of Winsock API referencing, would therefore cause detrimental performance loss.

```
; Provision Microsoft WinSock Extended APIs – (requires non-official sourcing of GUIDs values)
GuidAcceptEx GUID <0b5367df1h,0cbach,011cfh,{095h,0cah,000h,080h,05fh,048h,0a1h,092h}>
. . .
GuidWSARecvMsg GUID <0f689d7c8h,06f1fh,0436bh,{08ah,053h,0e5h,04fh,0e3h,051h,0c3h,022h}>
. . .
invoke WSAIoctl,DummySocket,SIO_GET_EXTENSION_FUNCTION_POINTER,addr  /
GuidAcceptEx,sizeof GUID, addr fnAcceptEx,sizeof fnAcceptEx,addr dwBytes,NULL,NULL
. . .
```

Codelet 8 Winsock extended function referencing

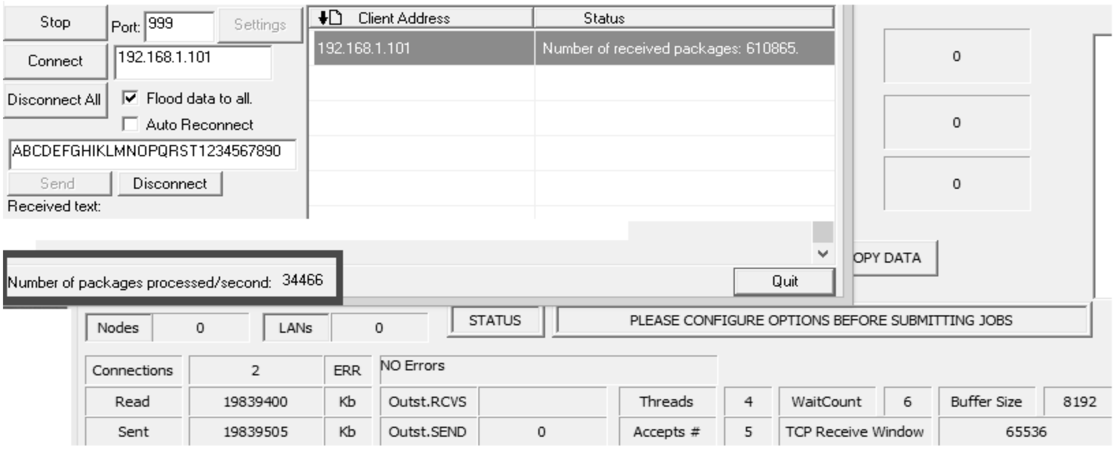
Each Winsock API function call from the extended range, is singularly aliased as static memory address pointers on artefact instantiation. The required parameter sets per API function, are likewise data typed in order to attain the overall utility.

A full list of the expanded Winsock API GUIDs currently remains unpublished by Microsoft. To add additional convolution, the OS supported release versions of the extended Winsock APIs are also erroneously indicated on the official developer website. It would seem that the periodic discontinuation of official support for a legacy Windows OS version warrants Microsoft to claim API support only for the next still supported version.

```
; Avoiding Stale Connections and Malicious Denial of Service
mov ebx, [esi].PendingAccepts
. . .
@@ParseStale:
.if (ebx != NULL)
invoke getsockopt,[ebx].sclient,SOL_SOCKET,SO_CONNECT_TIME,addr cTime, addr sTime
        .if (eax != SOCKET_ERROR) && (cTime != 0FFFFFFFFh) && (cTime > 20)
                invoke closesocket,[ebx].sclient
                mov [ebx].sclient, INVALID_SOCKET
        .endif
        mov ebx, [ebx].Next
        jmp @@ParseStale
.endif
. . .
```

Codelet 9 Sensing and eliminating stale connections

A noteworthy IOCP design feature within the prototype artefact, regards safeguards from notional malicious denial of service and stale connections. Principally, all pending and established connections would be subjected to time-out values. The connection is terminated and occupied resources released, should valid communications not be associable within the established timeframe. The current version of the artefact IOCP server was stress tested as a simple message echo implementation. Although not fully optimized using zero copy logic, the throughput achieved on an 802.11g wireless network exceeded 34 000 packets per second using a default MTU of 1500 bytes.



Port: 999

192.168.1.101

☑ Flood data to all.

□ Auto Reconnect

ABCDEFGHIKLMNOPQRST1234567890

Received text:

Client Address	Status
192.168.1.101	Number of received packages: 610865.

0

0

0

Number of packages processed/second: 34466

PLEASE CONFIGURE OPTIONS BEFORE SUBMITTING JOBS

Nodes	0	LANs	0						
Connections	2	ERR	NO Errors						
Read	19839400	Kb	Outst.RCVS		Threads	4	WaitCount	6	Buffer Size 8192
Sent	19839505	Kb	Outst.SEND	0	Accepts #	5	TCP Receive Window		65536

Figure 24 Stress testing IOCP module

Table 12 Appraisal of artefact resource extents

Artefact Component	Functional Execution Wall-clock Time (sec)	Process Memory Working Set (bytes)	Functional on Disk Size (bytes)	GUI on Disk Size (bytes)
System Profiler (Sample)	0,538	37 023 744	12 288	151 552
System Profiler (Dynamic link library)	0,023	4 784 128	12 288	-
Server	-	9 449 472	-	163 840
Client	-	4 718 592	32 768	-

The combined artefact's resource usage and physical footprint, as described in Table 12, is noted to be perceptibly diminutive. Per example the functional extents of the artefact system profiler, uses less than 5% the secondary storage requirement of the smallest functional profiling solution surveyed. Actual functional utility is achieved within an average reported wall-clock time of 0,023 seconds. The deployed artefact prototype, when compared with the smallest footprint non-deployed HPC or HTC solutions reviewed, reveals the artefact currently uses less than 0.5% of the secondary storage space. Granted that the artefact build is still in its infancy, the trending eventual build dimensions appear favourable. The resource extents of the artefact, therefore conforming to the ideal of rather maximizing resource availability towards actual workload utility consumption.

5.4 Limitations of Research

5.4.1 Generalizability Limitations

Due to the multi-dimensional nature of the research scope, the probability of reduced generalizability may be concluded for highly divergent research contexts. Per illustration, the range of literature engaged within the research action was by no means exhaustive for a particular scope context. The comprehensive problems experienced within a particular research area, could reveal counter arguments and exceptions to generalizability. Any claims to generalizability should only be considered within the exacting milieu of the described technical debt context. In retrospect, the researcher's ambition undoubtedly outstripped the latitude afforded by the research sponsor body. The innate nature of the research topic's derived subject material is clearly voluminous and on occasion exceedingly technical. As a result, the research narrative may have perceptually suffered to mention enough detail to satisfy the discerning reader. The potential to reproduce outcomes, by including the reference works, reviewed contemporary solution binaries and prototype design artefacts (i.e. available on accompanying DVD) are offered in mitigation of possible apprehension.

5.4.2 Design Limitations

The presented design artefact as holistic solution is incomplete and untested for use in real-world HPC and HTC hybridized enactments. Pointedly, the predominant characteristics of the design, currently only meet the defined minimal standards of an HPC solution. Although the initial ingredients of an HTC implementation is present, the scheduling, DFS and failover modules are conspicuously absent. What's more, the prototype artefact instantiation is imperfect at reflecting the design intent. Some preliminary enhancements suggested for incorporation include:

- The system profiler could benefit from added detection mechanisms for modern hardware features, such as random number generators and transactional synchronization extensions (e.g. lock elision)
- The IOCP module requires broad optimisation and the merger of Winsock extensions into the backend decision logic
- The overall solution’s binary executables require extended validation level code signing, to nullify administrative and security context issues experienced on newer Windows platforms that support Microsoft Authenticode and User Access Control

The lacking module components and real-world demonstration of hybridized utility is envisioned to be the subject of a future research action.

5.5 Chapter Summary

Chapter 5 establishes the design solution as an experimental artefact instantiation. The research endeavour’s major contexts are successively evaluated against the observed utility of the artefact. An outline regarding the functional responsibility of an individual design module, together with its link to a particular research context is conducted. In most instances, program codelets are presented to reinforce discussions. The deductive findings made are of importance on the subject of generalizability. To conclude the chapter, the foremost limitations incurred by the research action and its design construct was deliberated.

CHAPTER SIX
CONCLUSION

The research investigations directed within this paper initially chronicles and uncovers, the probable causes and outcomes of the technical debt phenomenon. Largely the research descriptive reinforces the link between technical debt, obsolescence and e-waste as an outflow. The connotations of technical debt should be of realistic concern for organisations and entities considering cloud adoption. Importantly the value proposition afforded by the move to cloud, necessitates assessment against the fact that significant loss could manifest from adoption. The value sought by cloud adoption, regards the leveraging of purported benefits in the creation of effective and efficient environments. The narrative of loss, labels amongst others the fiscal, ethical and functional impacts of non-holistic decision making. The currently retained IT assets that support needful processes, may explicitly require reconfiguration or be made obsolete. The assets comprising aspects of platform hardware, software and network infrastructure. According to the currently available global market data, indications are that entities overwhelmingly operate on Microsoft Windows platforms. Reasonably the potential for technical debt accumulation experienced by these entities may be greater.

In an effort to complement and align with the future value sought by cloud and data analytical solutions, this research paper proposes a design alternative that may delay the technical debt burden. Presumptively an innovative design artefact could facilitate the integration of major aspects of a contextualized technical debt platform and in so doing enable prolonged utility. The research design consequently calls for the creation of scalable HPC and HTC grid solutions. Not only could grid solutions factually reproduce most of the stated cloud adoption benefits, but importantly utilize existing IT assets as platform. The design discussions however expose several obstacles to grid realization. Notably the obstacles of heterogeneous platforms and abstraction are prominent. A comprehensive heterogeneity problem environment has questionably not been addressed in the survey literature. The role of abstraction in modern software development, furthermore resulting in difficulties surrounding the subjects of efficacy and performance. In response to these obstacles, the research endeavour takes on concrete assumptions and radical design patterns that embrace additional transformation effort to garner realization.

The immediate design outcomes and evidenced artefact instantiations promisingly exhibit several theoretically generalizable features and capabilities. Future research will endeavour to finalize the design model and demonstrate hybridized HPC and HTC workloads.

BIBLIOGRAPHY

Advanced Micro Devices, 2005. *AMD Processor Recognition,* Sunnyvale, California, United States: Advanced Micro Devices Inc.

Advanced Micro Devices, 2014. *Software Optimization Guide for AMD Family 15h Processors,* Sunnyvale, California, United States: AMD.

Akhter, S. & Roberts, J., 2006. *Multi-Core Programming: Increasing Performance through Software Multithreading.* 1st ed. Santa Clara, California, United States: Intel Press.

Ali, M. M., Southern, J., Strazdins, P. & Harding, B., 2014. *Application Level Fault Recovery: Using Fault-Tolerant Open MPI in a PDE Solver.* Phoenix, AZ, International Parallel & Distributed Processing Symposium Workshops - IEEE.

Alverson, G. et al., 1992. *Exploiting Heterogeneous Parallelism on a Multithreaded Multiprocessor.* Washington, D.C., USA, Proceedings of the International Conference on Supercomputing - ACM.

Berlin, K. et al., 2004. *Evaluating the Impact of Programming Language Features on the Performance of Parallel Applications on Cluster Architectures.* Heidelberg, Languages and Compilers for Parallel Computing.

Betz, . et al., 2015. *Sustainability Debt: A Metaphor to Support Sustainability Design Decisions.* Ottawa, Canada,
Fourth International Workshop on Requirements Engineering for Sustainable Systems (RE4SuSy).

Bezuidenhout, R., Davis, C. & Du Plooy-Cilliers, F., 2014. *Research Matters.* Claremont: Juta and Company Ltd.

Bidgoli, H., 2016. *Management Information Systems 6.* 6 ed. Boston: Cengage Learning.

Blevis, E., 2007. *Sustainable Interaction Design: Invention & Disposal, Renewal & Reuse.* San Jose, California, USA, Proceedings of the SIGCHI Conference on Human Factors in Computing Systems.

Botta, A., Donato, W. d., Persico, V. & Pescapé, A., 2016. Integration of Cloud computing and Internet of Things: A survey. *Future Generation Computer Systems - Elsevier,* 56(C), pp. 684-700.

Breivold, H. P. & Crnkovic, I., 2014. *Cloud Computing Education Strategies.* Klagenfurt, IEEE Software Engineering Education and Training.

Brooks, F. P., 1996. The Computer Scientist as Toolsmith II. *Communications of the ACM March 1996/Vol. 39, No. 3,* 1 March, 39(3), pp. 61-68.

Broquedis, F. et al., 2010. *Structuring the Execution of OpenMP Applications for Multicore Architectures.* Atlanta, GA, USA, IEEE International Symposium on Parallel & Distributed Processing (IPDPS).

Broquedis, F. et al., 2010. *hwloc: A Generic Framework for Managing Hardware Affinities in HPC Applications.* Pisa, Italy, 18th Euromicro International Conference on Parallel, Distributed and Network-based Processing - IEEE.

Brown, N. et al., 2010. *Managing Technical Debt in Software-Reliant Systems.* Santa Fe, New Mexico, USA, FoSER '10, ACM, pp. 47-52.

Bryman, A. et al., 2014. *Research Methodolgy: Business and Management Contexts.* 1st ed. Goodwood: Oxford University Press.

Buono, D., Matteis, T. D., Mencagli, G. & Vanneschi, M., 2014. *Optimizing Message-Passing on Multicore Architectures using Hardware Multi-Threading.* Turin, Italy, 22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing IEEE.

Buyya, R., Abramson, D. & Giddy, J., 2000. *Nimrod/G: AnArchitecture for a Resource Management and Scheduling System in a Global Computational Grid.* Beijing, China, Proceedings Fourth International Conference/Exhibition on High Performance Computing in the Asia-Pacific Region IEEE.

Chang, L. et al., 2014. *HAWQ: A Massively Parallel Processing SQL Engine in Hadoop.* Snowbird, UT, USA, Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data.

Chang, V. & Wills, G., 2015. A model to compare cloud and non-cloud storage of Big Data. *Future Generation Computer Systems,* Volume 57, p. 56–76.

Checkland, P., 1985. From Optimizing to Learning: A Development of Systems Thinking for the 1990s. *The Journal of Operational Research Society,* April, 36(9), pp. 757-767.

Che, D., Safran, M. & Peng, Z., 2013. From Big Data to Big Data Mining: Challenges, Issues and Opportunities. In: W. Winiwarter & W. Song, eds. *Database Systems for Advanced Applications.* Berlin: Springer, pp. 1-15.

Chen, H., Chiang, R. H. & Storey, V. C., 2012. Business Intelligence and Analytics: From Big Data to Big Impact. *MIS Quarterly,* 36(4), pp. 1165-1188.

Ciampa, M., 2015. *Security+ Guide to Network Security Fundementals.* 5th ed. Boston, MA: Cencage Learning.

Darlington, J. et al., 1993. *Parallel Programming Using Skeleton Functions.* Munich, Germany, Proceedings of the 5th International PARLE Conference on Parallel Architectures and Languages Europe.

Deelman, E., 2010. Grids and Clouds: Making Workflow Applications Work in Hetrogenous Distributed Environments. *The International Journal of High Performance Applications - Sage Journals,* 24(3), pp. 284-298.

Dhar, S. & Mazumder, S., 2014. Challenges and Best Practices for Enterprise Adoption of Big Data Technologies. *Journal of Information Technology Management,* 25(4), pp. 38-44.

Dobre, C. & Xhafa, F., 2014. Parallel Programming Paradigms and Frameworks in Big Data Era. *International Journal of Parallel Programming,* 43(5), pp. 710-738.

Doulkeridis, C. & Nørvåg, K., 2014. A survey of large-scale analytical query processing in MapReduce. *The VLDB Journal,* 23(DOI 10.1007/s00778-013-0319-9), pp. 355-380.

Drepper, U., 2007. *What Every Programmer Should Know About Memory,* Raleigh, North Carolina, United States: Red Hat Inc..

Dwivedy, M. & Mittal, R., 2010. Future trends in computer waste generation in India. *Waste Management,* 30(11), pp. 2265-2277.

Eklöv, D. & Hagersten, E., 2010. *StatStack: Efficient Modeling of LRU Caches.* New York, USA, International Symposium on Performance Analysis of Systems and Software - IEEE.

Ernst, N. A. et al., 2015. *Measure It? Manage It? Ignore It? Software Practitioners and Technical Debt.* Bergamo, Italy, ESEC/FSE'15, ACM.

Fischer, C., 2011. *The Information Systems Design Science Research Body Of Knowledge - A Citation Analysis In Recent Top-Journal Publications.* Brisbane, AIS Electronic Library, PACIS 2011 Proceedings, Paper 60, pp. 1-12.

Fitzpatrick, C., Hickey, S., Schischke, K. & Maher, P., 2014. Sustainable life cycle engineering of an integrated desktop PC; a small to medium enterprise perspective. *Journal of Cleaner Production,* Volume 74, pp. 155-160.

Fog, A., 2017. *Optimizing Subroutines in Assembly Language: An optimization guide for x86 platforms,* Lyngby: Technical University of Denmark.

Foster, I., Zhao, Y., Raicu, I. & Lu, S., 2008. *Cloud Computing and Grid Computing 360-Degree Compared.* Austin, IEEE .

García-Dorado, J. L. et al., 2013. High-Performance Network Traffic Processing Systems Using Commodity Hardware. *Lecture Notes in Computer Science: Data Traffic Monitoring and Analysis,* 7754(1), pp. 3-27.

Gartner Inc, 2016. *Gartner News Room.* [Online]
Available at: http://www.gartner.com/newsroom/id/3354117
[Accessed 28 June 2016].

Gepner, P. & Kowalik, M. F., 2006. *Multi-Core Processors: New Way to Achieve High System Performance.* Bialystok, Poland, International Symposium on Parallel Computing in Electrical Engineering - IEEE.

Gido, J. & Clements, J., 2014. *Successful Project Management.* 6th ed. Boston: Cengage Learning.

González, A., Aliagas, C. & Valero, M., 1995. *A Data Cache with Multiple Caching Strategies Tuned to Different Types of Locality.* Barcelona, Spain, International Conference on Supercomputing - ACM.

Gregor, S. & Jones, D., 2007. The Anatomy of a Design Theory. *Journal of the Association for Information Systems,* 8(5), pp. 312-225.

Grolinger, K. et al., 2014. *Challenges for MapReduce in Big Data.* Alaska, IEEE, pp. 182-189.

Grossman, M., Breternitz, M. & Sarkar, V., 2013. *HadoopCL: MapReduce on Distributed Heterogeneous Platforms through Seamless Integration of Hadoop and OpenCL.* Boston, IEEE.

Gupta, A., 2015. *Big Data Analysis Using Computaional Intelligence and Hadoop: A Study.* New Delhi, Computing for Sustainable Global Development.

Hashem, I. A. T. et al., 2015. The rise of "big data" on cloudcomputing: Review and open research issues. *Elsevier Information Systems,* Volume 47, pp. 98-115.

Herlihy, M., 2010. *Transactional Memory Today.* Berlin Heidelberg, International Conference on Distributed Computing and Internet Technology - Springer.

Herlihy, M. & Moss, J. E. B., 1993. *Transactional Memory: Architecture Support for Lock-Free Data Structures.* San Diego, CA, USA, International Symposium on Computer Architecture - IEEE.

Hevner, A. R., March, S. T. & Park, J., 2004. Design Science in Information Systems Research. *MIS Quaterly,* March, 28(1), pp. 75-105.

IBM Corporation, 2008. *IBM developerWorks.* [Online]
Available at: https://www.ibm.com/developerworks/library/l-devctrl-migration/index.html
[Accessed 28 June 2017].

Iivari, J., 2015. Distinguishing and Contrasting Two Strategies for Design Science Research. *European Journal of Information Systems,* 24(1), p. 107–115.

Intel Corporation, 2016. *Intel® 64 and IA-32 Architectures Optimization Reference Manual,* Santa Clara, California, United States: Intel.

Intel Corporation, 2017. *Product Brief: Intel® HPC Orchestrator,* Santa Clara, USA: Intel.

Jeong, H. & Lee, W., 2012. *Performance of SSE and AVX Instruction Sets.* Cairns, Proceedings of Science.
Jin, H.-W.et al., 2005. *Performance Evaluation of RDMA over IP: A Case Study with the Ammasso Gigabit Ethernet NIC.* Research Triangle Park, NC, USA, Workshop on High Performance Interconnects for Distributed Computing; In conjunction with HPDC-14.
Jones, A. & Ohlund, J., 2002. *Network Programming for Microsoft Windows.* 2nd ed. Redmond, Washington, USA: Microsoft Press.
Kalia, A., Kaminsky, M. & Andersen, D. G., 2016. *Design Guidelines for High Performance RDMA Systems.* Denver, CO, USA, Proceedings of USENIX Annual Technical Conference.
Katal, A., Wazid, M. & Goudar, R. H., 2013. *Big Data: issues, Challenges, Tools and Good Practices.* Noida, IEEE.
Kazempour, V., Fedorova, A. & Alagheband, P., 2008. *Performance Implications of Cache Affinity on Multicore Processors.* Las Palmas de Gran Canaria, Spain, European Conference on Parallel Processing - Springer.
Kennedy, K., Koelbel, C. & Schreiber, R., 2004. Defining and Measuring the Productivity of Programming Languages. *The International Journal of High Performance Computing Applications,* 18(4), pp. 441-448.
Kiczales, G. et al., 1997. *Aspect-Oriented Programming.* Finland, European Conference on Object-Oriented Programming.
Klinger, T., Tarr, P., Wagstrom, P. & Williams, C., 2011. *An Enterprise Perspective on Technical Debt.* Waikiki, Honolulu, Hawaii, ICSE ’11, ACM.
Kozierok, C., 2005. *The TCP/IP Guide: A Comprehensive, Illustrated Internet Protocols Reference.* 1st ed. San Francisco: No Starch Press.
Kraska, T., 2013. Finding the Needle in the Big Data Systems Haystack. *IEEE Internet Computing*, 15 January/February, pp. 84-86.
Kuechler, B. & Vaishnavi, V., 2008. *Theory Development in Design Science Research: Anatomy of a Research Project.* Georgia, Proceedings of the Third International Conference on Design Science Research in Information Systems and Technology.
Kuo, S., 2017. *Intel Developer Zone - WhitePapers.* [Online]
Available at: https://software.intel.com/en-us/articles/intel-64-architecture-processor-topology-enumeration
[Accessed 15 2 2017].
Kusswurm, D., 2014. *Modern X86 Assembly Language Programming 32-bit, 64-bit, SSE, and AVX.* 1st ed. New York: Apress.
Lee, E., 2006. The Problem with Threads. *Computer - IEEE Computer Society,* 39(5), pp. 33-42.
Leetaru, K., 2016. *Forbes Media, Tech / #Big Data.* [Online]
Available at: https://www.forbes.com/sites/kalevleetaru/2016/08/24/how-the-cloud-stands-to-reshape-academic-computing/#29d408355372
[Accessed 7 3 2018].
Lee, V. W. et al., 2010. *Debunking the 100X GPU vs. CPU Myth: An Evaluation of Throughput Computing on CPU and GPU.* Saint-Malo, France, Proceedings of the 37th annual international symposium on Computer architecture - ACM.
Li, K. & Hudak, P., 1989. Memory Coherence in Shared Virtual Memory Systems. *Transactions on Computer Systems - ACM,* 7(4), pp. 321-359.
Liu, J., Wu, J. & Panda, D. K., 2004. High Performance RDMA-Based MPI Implementation over InfiniBand. *International Journal of Parallel Programming,* 32(3), pp. 167-198.
Majo, Z. & Gross, T. R., 2011. *Memory Management in NUMA Multicore Systems: Trapped between Cache Contention and Interconnect Overhead.* San Jose, California, USA, The International Symposium on Memory Management - ACM.
Majo, Z. & Gross, T. R., 2013. *(Mis)understanding the NUMA Memory System Performance of Multithreaded Workloads.* Portland, Oregon, USA, International Symposium on Workload Characterization - IEEE.
Majo, Z. & Gross, T. R., 2015. *A Library for Portable and Composable Data Locality Optimizations for NUMA Systems.* New York, USA, Proceedings of the Symposium on Principles and Practice of Parallel Programming - ACM.
Mantripragada, K., Binotto, A. & Tizzei, L. P., 2015. *A Self-adaptive Auto-scaling Method for Scientific Applications on HPC Environments and Clouds.* Amsterdam, Netherlands, ADAPT Workshop proceedings.
Martin, A., Britoy, A. & Fetzer, C., 2014. *Elastic and Secure Energy Forecasting in Cloud Environments.* London, IEEE.
Martini, A., Bosch, J. & Chaudron, M., 2014. *Architecture Technical Debt: Understanding Causes and a Qualitative Model.* Verona, Italy , 2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications, IEEE.

Masud, M. A. H., Yong, J. & Huang, X., 2012. *Cloud Computing for Higher Education: A Roadmap.* Wuhan, China, International Conference on Computer Supported Cooperative Work in Design - IEEE.

Mclean, I. & Thomas, O., 2010. *Configuring Windows 7 - MCTS Self-Paced Training Kit (Exam70-680).* 1 ed. Redmond, Washington: Microsoft Press.

Microsoft Corporation, 2001. *Winsock Direct: The Value of System Area Networks.* [Online]
Available at: https://msdn.microsoft.com/en-us/library/bb742608.aspx
[Accessed 4 10 2017].

Microsoft Corporation, 2003. *Microsoft Technet.* [Online]
Available at: https://technet.microsoft.com/en-us/library/cc776246(v=ws.10).aspx
[Accessed 28 June 2017].

Microsoft Corporation, 2013. *Understanding Node Metrics and Properties in HPC Cluster Manager.* [Online]
Available at: https://technet.microsoft.com/en-us/library/hh301098(v=ws.11).aspx
[Accessed 17 8 2017].

Microsoft Corporation, 2016. *Microsoft Volume Licensing.* [Online]
Available at: http://www.microsoftvolumelicensing.com
[Accessed 31 March 2017].

Microsoft Corporation, 2016. *System Requirements for HPC Pack 2016.* [Online]
Available at: https://technet.microsoft.com/en-us/library/mt791806(v=ws.11).aspx
[Accessed 12 7 2017].

Microsoft Corporation, 2017. *High-performance Windows Sockets Applications.* [Online]
Available at: https://msdn.microsoft.com/en-us/library/windows/desktop/ms738551%28v=vs.85%29.aspx?f=255&MSPPError=-2147217396
[Accessed 31 10 2017].

Microsoft Corporation, 2017. *Microsoft.* [Online]
Available at: https://www.microsoft.com/en-us/Licensing/licensing-programs/licensing-programs.aspx
[Accessed 31 March 31].

Microsoft Corporation, 2017. *Microsoft Support.* [Online]
Available at: https://support.microsoft.com
[Accessed 31 March 31].

Montero, R. S., Moreno-Vozmediano, R. & Llorente, I. M., 2011. An elasticity model for High Throughput Computing clusters. *Journal of Parallel and Distributed Computing,* Volume 71, pp. 750-757.

Moreton-Fernandez, A., Ortega-Arranz, H. & Gonzalez-Escribano, A., 2017. Controllers: An abstraction to ease the use of hardware accelerators. *The International Journal of High Performance Computing Applications.*

Murray, D., Koziniec, T., Lee, K. & Dixon, M., 2012. *Large MTUs and Internet Performance.* Belgrade, Serbia, International Conference on High Performance Switching and Routing - IEEE.

Nakajima, J. & Pallipadi, V., 2002. *Enhancements for hyper-threading technology in the operating system: seeking the optimal scheduling.* Boston, MA, USA, Proceedings of the 2nd conference on Industrial Experiences with Systems Software - IEEE, USENIX Association Berkeley.

Nelson, J. et al., 2015. *Latency-Tolerant Software Distributed Shared Memory.* Santa Clara, CA, USA , Proceedings of the USENIX Annual Technical Conference - ACM.

Net Applications, 2017. *Netmarketshare.* [Online]
Available at: http://www.netmarketshare.com/operating-system-market-share.aspx
[Accessed 27 March 2017].

Neuman, W., 2011. *Social Research Methods Qualitative and Quantitative Approaches.* 7th ed. Boston: Pearson Education, Inc. publishing as Allyn & Bacon.

Newburn, C. J. et al., 2011. *Intel's Array Building Blocks: A Retargetable, Dynamic Compiler and Embedded Language.* Chamonix, IEEE: International Symposium on Code Generation and Optimization.

Open Science Grid, 2016. *Open Science Grid Wiki.* [Online]
Available at: https://twiki.opensciencegrid.org/bin/view/Documentation/WhatIsOSG
[Accessed 21 7 2017].

Ostrowski, L. & Helfert, M., 2012. Design Science Evaluation – Example of Experimental Design. *Journal of Emerging Trends in Computing and Information Sciences,* 3(9), pp. 253-262.

Ousterhout, J. K., 1982. *Scheduling Techniques for Concurrent Systems.* New York, Proceedings of the 3rd International Conference on Distributed Computing Systems - IEEE.

Peffers, K., Rothenberger, M., Tuunanen, T. & Vaezi, R., 2012. Design Science Research Evaluation. *Design Science Research in Information Systems. Advances in Theory and Practice. ,* 7286(1), pp. 398-410.

Peffers, K., Tuunanen, T., Rothenberger, M. A. & Chatterjee, S., 2007. A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems,* 24(3), pp. 45-78.

Petridis, N. E., Stiakakis, E., Petridis, K. & Dey, P., 2016. Estimation of computer waste quantities using forecasting techniques. *Journal of Cleaner Production,* 112(4), pp. 3072-3085.

Prakash, P. et al., 2013. *Jumbo Frames or Not: That is the Question!,* West Lafayette, Indiana, USA: Computer Science Technical Reports, Paper 1770 - Purdue University.

Prat, N., Comyn-Wattiau, I. & Akoka, J., 2014. *Artifact Evaluation in Information Systems Design-Science Research - a Holistic View.* Chengdu, AIS Electronic Library, PACIS 2014 Proceedings, Paper 23, pp. 1-16.

Pries-Heje, J., Baskerville, R. & Venable, J. R., 2008. *Strategies for Design Science Research Evaluation.* Galway, Ireland, 16th European Conference on Information Systems.

Regnier, G. et al., 2004. TCP Onloading for Data Center Servers. *Computer - IEEE,* 37(11), pp. 48-58.

Remy, C. & Huang, E., 2015. *Addressing the Obsolescence of End-User Devices: Approaches from the Field of Sustainable HCI.* Zürich, Switzerland, Springer, pp. 257-267.

Reyes-Ortiz, J. L., Oneto, L. & Anguita, D., 2015. Big Data Analytics in the Cloud: Spark on Hadoop vs MPI/OpenMP on Beowulf. *Procedia Computer Science,* Volume 53, pp. 121-130.

Robinson, B. H., 2009. E-waste: An assessment of global production and environmental impacts. *Science of the Total Environment,* Volume 408, p. 183–191.

Romanow, A. & Bailey, S., 2003. *An Overview of RDMA over IP.* CERN, Geneva, Switzerland, Proceedings of the Workshop on Protocols for Fast Long-Distance Networks - The Internet Society.

Rossbach, C. J. et al., 2013. *Dandelion: a Compiler and Runtime for Heterogeneous Systems.* Farmington, Pennsylvania, USA, Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles.

Sabharwal, M., Agrawal, A. & Metri, G., 2013. Enabling Green IT through Energy-Aware Software. *IT Professional - IEEE,* 15(1), pp. 19-27.

Sandberg, A., Eklöv, D. & Hagersten, E., 2010. *Reducing Cache Pollution Through Detection and Elimination of Non-Temporal Memory Accesses.* New Orleans, Louisiana, USA, International Conference for High Performance Computing, Networking, Storage and Analysis - IEEE / ACM.

Satzinger, J., Jackson, R. & Burd, S., 2012. *Introduction to Systems Analysis and Design: An agile, iterative approach.* 6th ed. Toronto: Cengage.

Saxena, S., Sharma, S. & Sharma, N., 2016. Parallel Image Processing Techniques, Benefits and Limitations. *Research Journal of Applied Sciences, Engineering and Technology,* 12(2), pp. 223-238.

Schiesser, R., 2010. *IT Systems Management.* 2nd ed. Boston, MA: Pearson Education, Inc.

Seshadri, V., Mutlu, O., Kozuch, M. A. & Mowry, T. C., 2012. *The Evicted-Address Filter: A Unified Mechanism to Address Both Cache Pollution and Thrashing.* Minneapolis, Minnesota, USA, International Conference on Parallel Architectures and Compilation Techniques - IEEE / ACM.

Sharmilarani, D., Vinothini, K., Ramya, V. & Shobika, R., 2017. An Efficient Resource Aware Scheduling Algorithm for Mapreduce Clusters. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology,* 2(2), pp. 517-523.

Shaw, M., 1980. The impact of abstraction concerns on modern programming languages. *Proceedings of the IEEE,* 68(9), pp. 1119 - 1130.

Shvachko, K., Kuang, H., Radia, S. & Chansler, R., 2010. *The Hadoop Distributed File System.* Incline Village, NV, USA, The Symposium on Mass Storage Systems and Technologies - IEEE.

Simon, H. A., 1996. *The Sciences of the Artificial.* 3rd ed. Massachusetts(Cambridge): MIT Press.

Smith, J. & Goodman, J., 1985. Instruction Cache Replacement Policies and Organizations. *IEEE Transactions on Computers,* 34(1), pp. 234-241.

Speight, E., Shafi, H. & Bennett, J. K., 2000. *WSDLite: a Lightweight Alternative to Windows Sockets Direct Path.* Seattle, Washington, USA, Proceedings of the conference on USENIX Windows Systems - ACM.

Statista Inc, 2016. *Satista - The statistics portal.* [Online]
Available at: https://www.statista.com/statistics/272595/global-shipments-forecast-for-tablets-laptops-and-desktop-pcs/
[Accessed 27 March 2017].

Statistic Brain Research Institute, 2016. *Statistic Brain.* [Online]
Available at: http://www.statisticbrain.com/computer-sales-statistics/
[Accessed 27 March 2017].

Stratton, J. A., Stone, S. S. & Wen-mei, H. W., 2008. *MCUDA: An Effcient Implementation of CUDA Kernels on Multi-cores.* Edmonton, AB, Canada, International Workshop on Languages and Compilers for Parallel Computing.

Suh, G., Rudolph, L. & S.Devadas, 2004. Dynamic Partitioning of Shared Cache Memory. *The Journal of Supercomputing,* 28(1), pp. 7-26.

The WinSock Standard Group, 1997. *Windows Sockets 2 Application Programming Interface,* s.l.: The WinSock Group .

Tom, E., Aurum, A. & Vidgen, R., 2013. An exploration of technical debt. *The Journal of Systems and Software,* Volume 86, p. 1498– 1516.

University of Wisconsin-Madison, 2017. *HTCondorTM Version 8.7.2 Manual.* [Online]
Available at: http://research.cs.wisc.edu/htcondor/manual/v8.7/index.html
[Accessed 28 7 2017].

Venable, J., Pries-Heje, J. & Baskerville, R., 2012. *A Comprehensive Framework for Evaluation in Design Science Research.* Las Vegas, NV, USA, International Conference on Design Science Research in Information Systems - Springer.

Venable, J., Pries-Heje, J. & Baskerville, R., 2016. FEDS: a Framework for Evaluation in Design Science Research. *European Journal of Information Systems,* 25(1), p. 77–89.

Ward, J. & Peppard, J., 2002. *Strategic Planning for Information Systems.* 3rd ed. New York: Wiley.

White, T., 2012. *Hadoop: The Definitive Guide.* 3rd ed. Beijing, Cambridge, Farnham, Köln, Sebastopol, Tokyo: O'Reilly, Yahoo! Press.

Widmer, R. et al., 2005. Global perspectives on e-waste. *Environmental Impact Assessment Review,* Volume 25, p. 436–458.

Wieringa, R., 2009. *Design Science as Nested Problem Solving.* Philadelphia, Pennsylvania, Proceedings of the International Conference on Design Science Research in Information Systems and Technology.

Wieringa, R., 2016. *Design Science Methodology MIKS,* Enschede, Netherlands: University of Twente.

Xie, J. et al., 2012. *Improving MapReduce Performance through Data Placement in Heterogeneous Hadoop Clusters.* Atlanta, USA, Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW) IEEE.

Youseff, L., Butrico, M. & Silva, D. D., 2008. *Toward a Unified Ontology of Cloud Computing.* Austin, TX, USA, Grid Computing Environments Workshop IEEE.

Yu, J., Williams, E., Ju, M. & Yang, Y., 2010. Forecasting Global Generation of Obsolete Personal Computers. *Environmental Science & Technology,* 44(9), pp. 3232-3237.

Zhang, W., Shi, D. & Li, L., 2012. The Design of the Underlying Network Communication Module Based on IOCP. *Communications and Information Processing: Communications in Computer and Information Science,* 289(2), pp. 17-24.

Zhao, D. et al., 2014. *FusionFS: Towards Supporting Data-Intensive Scientific Applications on Extreme-Scale High-Performance Computing Systems.* Washington, USA, IEEE International Conference on Big Data (Big Data) .

APPENDICES

APPENDIX A: DETAILED COMPARISON OF SURVEYED SYSTEM PROFILERS

DATA POINT	OpenMPI hwloc / nwloc v2.0.1 Windows	CPUID™ CPU-Z v1.84.0	REALiX™ HWiNFO V5.74	OpenHardware Monitor v.0.8b	Apache® Hadoop YARN v2.8.0 Node Manager*
CENTRAL PROCESSING UNIT					
Brand	Yes	Yes	Yes	Yes	No
Run-time Speed	No	Yes	Yes	Yes	Yes
Instruction Set	No	Yes	Yes	Partial	No
Model & Family	Yes	Yes	Yes	Yes	No
Packages	Yes	Yes	Yes	Yes	Yes
Cores	Yes	Yes	Yes	Yes	Yes
CPU Threads	Yes	Yes	Yes	Yes	Yes
CPU Architecture	Yes	Yes	Yes	Yes	No
NUMA Topology	Yes	Yes	Yes	Partial	Yes
APIC Binding	Yes	No	Yes	Yes	No
APIC Type	No	Yes	Yes	Partial	No
Clock Type	No	Yes	Yes	Partial	No
CACHE TOPOLOGY					
L1 Instruction Size	Yes	Yes	Yes	Partial	No
L1 Data Size	Yes	Yes	Yes	Partial	No
L2 Size	Yes	Yes	Yes	Partial	No
L3 Size	Yes	Yes	Yes	Partial	No
Associativity	Yes	Yes	Yes	Partial	No
NUMA Topology	Yes	Yes	Yes	Partial	No
RANDOM ACCESS MEMORY					
Paging Size	No	No	Yes	No	No
Physical Size	Yes	Yes	Yes	Yes	Yes
Available Size	No	No	Yes	Yes	Yes
Process Maximum	No	No	No	No	Yes
Cache Line Size	Yes	Yes	Yes	No	No
SECONDARY STORAGE					
Disk Size	No	Partial	Yes	Yes	Yes
Disk Free	No	No	No	Yes	Yes
Process Start	No	No	No	No	Yes
OPERATING SYSTEM					
Version & Edition	Partial	Yes	Yes	Yes	Yes
Service Pack	No	Yes	Yes	No	No
Suite	No	No	No	No	No
OS Architecture	Yes	Yes	Yes	Yes	No
OS Threads	No	No	Yes	No	Yes
Security Context	Partial	Partial	Partial	Partial	Partial
NETWORK CONFIGURATION					
Socket API	No	No	No	No	No
Routing	No	No	No	No	Yes
Protocol Stack	No	No	No	No	Partial
Addressing	Partial	No	Partial	No	Yes
Local Latency	No	No	Partial	No	Yes
Path MTU Size	No	No	No	No	No

APPENDIX B: DESCRIPTIVE OF SYSTEM PROFILING DATA POINTS

B1 Central Processing Unit

Brand – The CPU brand is vital in determining the individual features and functions used for later platform profiling. To accurately reflect the hardware reality, CPU manufactures often publish prescriptive processes and mechanisms to adhere to.

Run-time Speed – The reported CPU speed may be different to manufacturer's factory release speed, due to aftermarket configuration or speed stepping. The difference between run-time and manufacturer intended speed, is frequently of interest. A potential bottle-neck in performance.

Instruction Set – The detection of instruction set features provided by the CPU, are important for parallelism opportunities. Especially SIMD and MIMD instructions are regarded as advantages. The detection of virtual machine extensions or speciality features, may indicate computational environmental constraints or prospects.

Model & Family – The model and family information of a processor may designate its intended computer form-factor, such as desktop, server or mobile platform. The information could potentially further be used to circumvent known CPU microcode issues.

Packages – The number of physical CPU dies supported. The more the better. Whether the CPU dies enable interaction, would be of programmatic concern.

Cores – The number of physical CPU processors supported per die. The number of physical hardware threads onto which software threads may be associated.

CPU Threads – The number of logical processors supported per CPU core. Either the same number as CPU cores or possibly double that number, dependant on hyper threading. The total number of CPU threads, should typically match the OS detected threads. A potential bottle-neck in performance.

CPU Architecture – The instruction set, maximum theoretical addressable memory and register width supported by the CPU, meaningfully being either x86 or x64. The OS may or may not have equivalent support.

NUMA Topology – The logical and/or physical processor topology could be different. The addressable CPU constituents may be combined as autonomous entities, although sharing the same or different physical underlying platform.

APIC Binding – The CPU addressable units or components, are enumerated and referenced using the advanced programmable interrupt controller. Of initial interest would be the bounded addressable CPU unit of the running software process. Important information for shifting software threads across physical processors (affinity) or setting software thread priority.

APIC Type – The advanced programmable interrupt controller type, indicates extensibility of the CPU's functional addressability. This would be important for NUMA and CPU cache detection scenarios.

Clock Type – The precision of the exposed platform timings, as measured in clock cycles per second, could be important for programmatic synchronization.

B2 Cache Topology

L1 Instruction & Data Size – The available storage at lowest level to the processor. The more the better. Valuable information for planning a fine-grained caching strategy. Also, of interest would be whether this storage is shared by other logical processors. A potential bottle-neck in performance.

L2 & L3 Size – The level 2 and 3 caches, are normally unified caches that do not discern between data or instructions. The more the better. Valuable information for planning any coarse-grained caching strategy. Also, of interest would be whether this storage is shared by other physical processors. A potential bottle-neck in performance.

Associativity – The cache associativity or mapping of cache entries to RAM, may be used in optimizing a fine-grained caching strategy.

NUMA Topology – As CPUs can be logically combined to form autonomous entities, the exposed caching structure could differ radically from the detected physical CPU package topology.

B3 Random Access Memory

Paging Size – The unit data size for virtual memory management. The page size value is stereotypically a variable in an I/O or cache management strategy.

Physical Size – The size of the physical RAM installed. Running processes share this storage area. A potential bottleneck in performance.

Available Size – The amount of free RAM. A process could use this information in determining the current primary storage load. A potential bottleneck in performance, as virtual memory paging could initiate when physical RAM storage is depleted.

Process Maximum – The maximum addressable memory a process could claim. The value includes the configured physical RAM and extending virtual storage areas.

Cache Line Size – The size of a cache block. Essentially a replica of line size value, from the corresponding are in RAM. May be used in devising of a fine-grained caching strategy.

B4 Secondary Storage

Disk Size – The secondary storage size. Typically a hard disk, network or solid state drive.

Disk Free – The amount of free hard disk space. The available secondary storage available to the process that initiated system profiling, could be of interest in assigning computational workload. The value is contextualized to the storage quota limit of the user initiating the profiling process.

Process Start – A process is initiated from within an executable file. The location of the file may create a reference point for storage planning.

B5 Operating System

Version & Edition – The version and edition of the OS, is vital information in determining holistic API compatibility for cross platform integration.

Service Pack – The service pack exposed by the OS might be important data regarding the security stance and general functionality provided.

Suite – Besides the kernel, an accompanying suite mask and type mask, can identify the additional OS components installed. The suite variable may well define the OS class. Some examples of OS classes, amongst others include enterprise server, data centre server, blade server, back office server, embedded or workstation.

OS Architecture – The hardware architecture supported by the OS kernel. The OS architecture and the CPU instruction set architecture, should ideally match.

OS Threads – The OS utilizes logically addressable CPU units as processing resources. Importantly, an obvious association between the hardware detected and the OS reported processing units could be difficult to make. The factors causing the phenomenon ranging from misconfiguration, license enforcement, NUMA implementations, virtual machine instantiation, manufacturer provisioning or hyper-threading scenarios.

Security Context – A process normally executes within the instantiated security context of a user. The privilege level of the user would be a key parameter for realizing I/O operations.

B6 Network Configuration

Socket API – The OS API available for socket operations, could expose appealing features that may enhance network performance and throughput.

Routing – Knowledge of the inter-network connection points can aid in LAN isolation and functional network topology planning. Plausibly, reduced latency and higher bandwidth environments are shared by LAN participants within an isolated or reduced broadcast domain. The information could be used to beneficially derive grid node roles within a LAN.

Protocol Stack – The information regarding the network protocols supported, can provide for flexibility and utility of network communications.

Addressing – Logical and physical network address resolution, may assist in LAN isolation and network topology planning. The domain administrative context of individual grid participants, could potentially be concluded.

Local Latency – The latency characteristics within an isolated LAN, can be used to embed situational efficiencies within grid node roles.

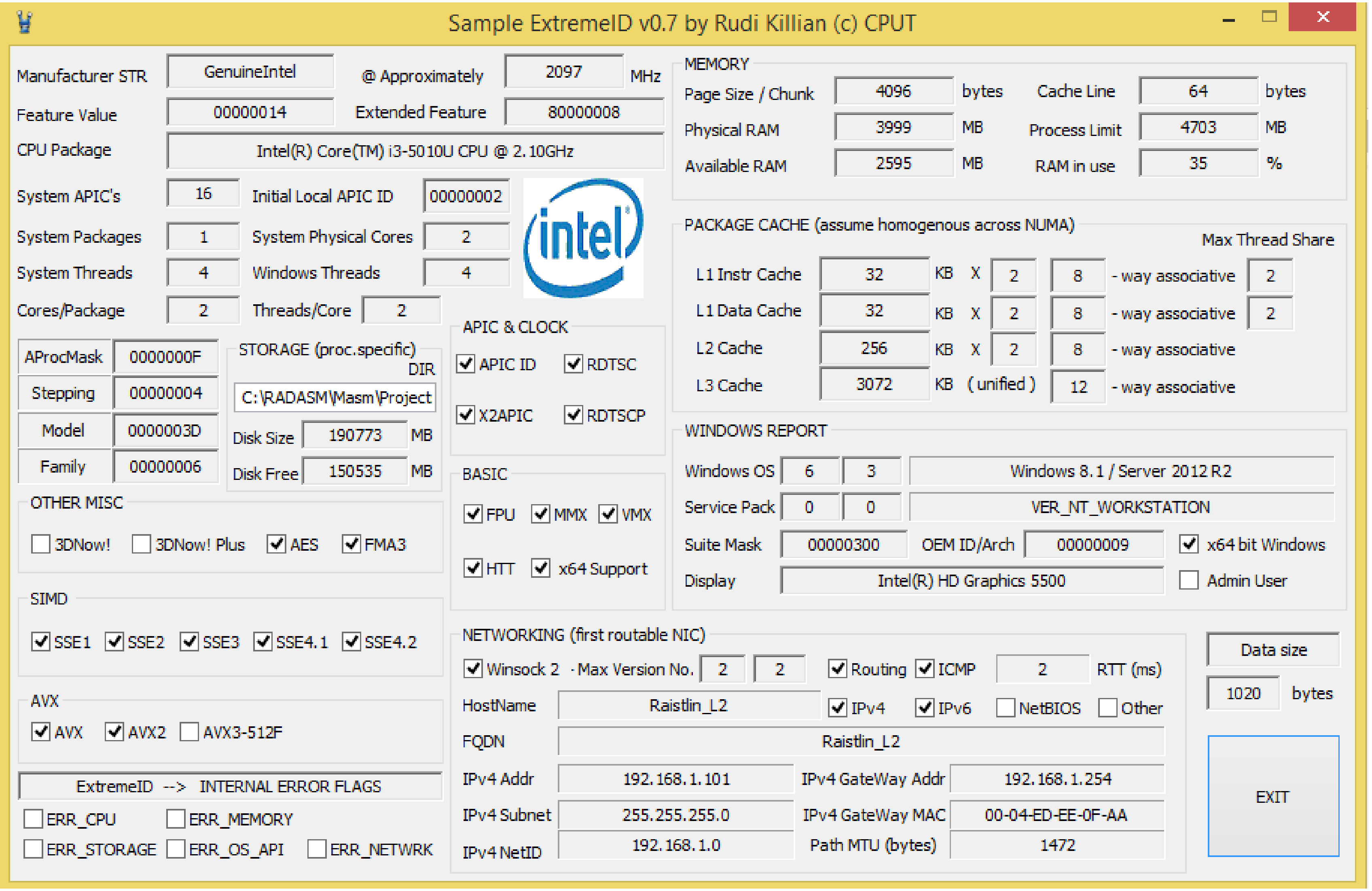
Path MTU Size – The minimum aggregate size of the maximum transmission unit over a network path. Used to advantage network I/O buffers and jumbo frames.

APPENDIX C: ARTEFACT SUNDRIES

C1 Sample Data Structure for Static System Profiling

```
PackageCount          dd ?             ; CPU Package count
TotalAPICs            dd ?             ; Logical addressable "Processors" / number of APIC's
TotalCores            dd ?             ; Total reported Cores (AMD might report more than actual)
TotalThreads          dd ?             ; Total reported Logical Threads
CoresperPackage       dd ?             ; Number of Cores per Package
ThreadsperCore        dd ?             ; Number of Threads per Core
LocalApicId           dd ?             ; Default APIC ID
;------------------------------------------------------------------------------------------------------------------
Stepping              dd ?             ; Speed step Technology
Model                 dd ?             ; CPU Model number
Family                dd ?             ; CPU Family Number
InputValue            dd ?             ; Max function level supported
ExtendedValue         dd ?             ; Max extended feature supported
CpuSpeed              dd ?             ; Approx. CPU Speed in Mhz, timed algorithm reported
;------------------------------------------------------------------------------------------------------------------
OSCores               dd ?             ; OS Reported Core count
OSArchitecture        dd ?             ; Processor architecture of the installed operating system
OSPagesize            dd ?             ; Page size /granularity of page protection/commitment
CacheLine             dd ?             ; Cache Line size in bytes
AProcMask             dd ?             ; Active Processor Mask
PhysicalRAM           dd ?             ; Physical Installed RAM
AvailableRAM          dd ?             ; Available Installed RAM
ApplimitRAM           dd ?             ; Max Limit for application use
MemLoad               dd ?             ; Approximate percentage of physical memory that is in use
OSMajor               dd ?             ; OS Major version
OSMinor               dd ?             ; OS Minor version
OSSrvPackMaj          dd ?             ; Service Pack Major
OSSrvPackMin          dd ?             ; Service Pack Minor
OSSuiteMask           dd ?             ; Bit mask identifying the product suites on the system
OSProdType            dd ?             ; OS Product Type ex. WRKST, SERVER, DOMAINCNTLR
;------------------------------------------------------------------------------------------------------------------
L1_D_Cache            dd ?             ; L1 DATA cache size
L1_D_Share            dd ?             ; L1 DATA cache - number of threads sharing
L1_D_Assoc            dd ?             ; L1 DATA Associativity
L1_I_Cache            dd ?             ; L1 INSTRUCTION cache size
L1_I_Share            dd ?             ; L1 INSTRUCTION cache - number of threads sharing
L1_I_Assoc            dd ?             ; L1 INSTRUCTION Associativity
L2_Cache              dd ?             ; L2 Cache size
L2_Assoc              dd ?             ; L2 Associativity
L3_Cache              dd ?             ; L3 Cache size
L3_Assoc              dd ?             ; L3 Associativity
;------------------------------------------------------------------------------------------------------------------
DiskSize              dd ?             ; Process Storage Size (MB)
DiskFree              dd ?             ; Process Storage Free (MB)
;------------------------------------------------------------------------------------------------------------------
wsa_MaxMajor          dd ?             ; Winsock Maximum Major version supported
wsa_MaxMinor          dd ?             ; Winsock Maximum Minor version supported
wsa_IPv4Address       dd ?             ; IPv4 Local NIC address in Network format
wsa_IPv4Subnet        dd ?             ; IPv4 Local Subnet Mask in Network format
wsa_IPv4Gateway       dd ?             ; IPv4 Local Gateway address in Network format
wsa_IPv4NetID         dd ?             ; IPv4 Network ID address in Network format
wsa_MTU               dd ?             ; Maximum Transmission Unit size in bytes
wsa_RTT               dd ?             ; Round Trip Time (on MTU frame size)
wsa_GatewayMAC        db 18  dup (?)   ; Gateway MAC address (LAN unique isolator)
wsa_IPv4Address_R     db 16  dup (?)   ; IPV4 Local NIC address human readable - char limit 15
wsa_IPv4Subnet_R      db 16  dup (?)   ; IPV4 Local Subnet human readable - char limit 15
wsa_IPv4Gateway_R     db 16  dup (?)   ; IPV4 Local Gateway human readable - char limit 15
wsa_IPv4NetID_R       db 16  dup (?)   ; IPV4 Network ID human readable - char limit 15
;------------------------------------------------------------------------------------------------------------------
OS_VERSION            db 35  dup (?)   ; Windows OS Version Name - char limit 34
OS_TYPE               db 25  dup (?)   ; Windows OS Type Workstation / Server - char limit 24
Display_ADAPTER       db 129 dup (?)   ; First Primary Display Description attached to Desktop
OS_PROCPATH           db 261 dup (?)   ; Process Executable Path (*possibly truncated)
. . .
```

C2 Screenshot Output of System Profiling Module



Sample ExtremeID v0.7 by Rudi Killian (c) CPUT

Manufacturer STR: GenuineIntel @ Approximately 2097 MHz
Feature Value: 00000014 Extended Feature 80000008
CPU Package: Intel(R) Core(TM) i3-5010U CPU @ 2.10GHz
System APIC's: 16 Initial Local APIC ID: 00000002
System Packages: 1 System Physical Cores: 2
System Threads: 4 Windows Threads: 4
Cores/Package: 2 Threads/Core: 2

AProcMask: 0000000F
Stepping: 00000004
Model: 0000003D
Family: 00000006

STORAGE (proc.specific)
DIR: C:\RADASM\Masm\Project
Disk Size: 190773 MB
Disk Free: 150535 MB

OTHER MISC
☐ 3DNow! ☐ 3DNow! Plus ☑ AES ☑ FMA3

SIMD
☑ SSE1 ☑ SSE2 ☑ SSE3 ☑ SSE4.1 ☑ SSE4.2

AVX
☑ AVX ☑ AVX2 ☐ AVX3-512F

ExtremeID --> INTERNAL ERROR FLAGS
☐ ERR_CPU ☐ ERR_MEMORY
☐ ERR_STORAGE ☐ ERR_OS_API ☐ ERR_NETWRK

APIC & CLOCK
☑ APIC ID ☑ RDTSC
☑ X2APIC ☑ RDTSCP

BASIC
☑ FPU ☑ MMX ☑ VMX
☑ HTT ☑ x64 Support

MEMORY
Page Size / Chunk: 4096 bytes Cache Line: 64 bytes
Physical RAM: 3999 MB Process Limit: 4703 MB
Available RAM: 2595 MB RAM in use: 35 %

PACKAGE CACHE (assume homogenous across NUMA)

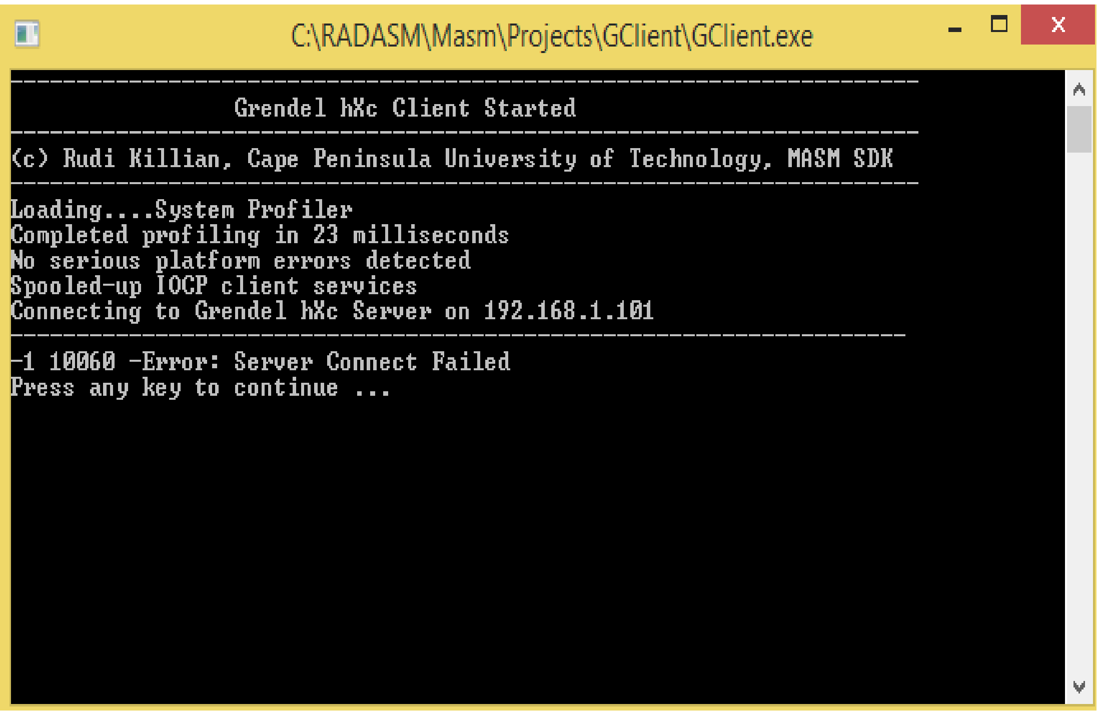
							Max Thread Share
L1 Instr Cache	32	KB	X	2	8	- way associative	2
L1 Data Cache	32	KB	X	2	8	- way associative	2
L2 Cache	256	KB	X	2	8	- way associative	
L3 Cache	3072	KB	(unified)		12	- way associative	

WINDOWS REPORT
Windows OS: 6 3 Windows 8.1 / Server 2012 R2
Service Pack: 0 0 VER_NT_WORKSTATION
Suite Mask: 00000300 OEM ID/Arch: 00000009 ☑ x64 bit Windows
Display: Intel(R) HD Graphics 5500 ☐ Admin User

NETWORKING (first routable NIC)
☑ Winsock 2 · Max Version No. 2 2 ☑ Routing ☑ ICMP 2 RTT (ms)
HostName: Raistlin_L2 ☑ IPv4 ☑ IPv6 ☐ NetBIOS ☐ Other
FQDN: Raistlin_L2
IPv4 Addr: 192.168.1.101 IPv4 GateWay Addr: 192.168.1.254
IPv4 Subnet: 255.255.255.0 IPv4 GateWay MAC: 00-04-ED-EE-0F-AA
IPv4 NetID: 192.168.1.0 Path MTU (bytes): 1472

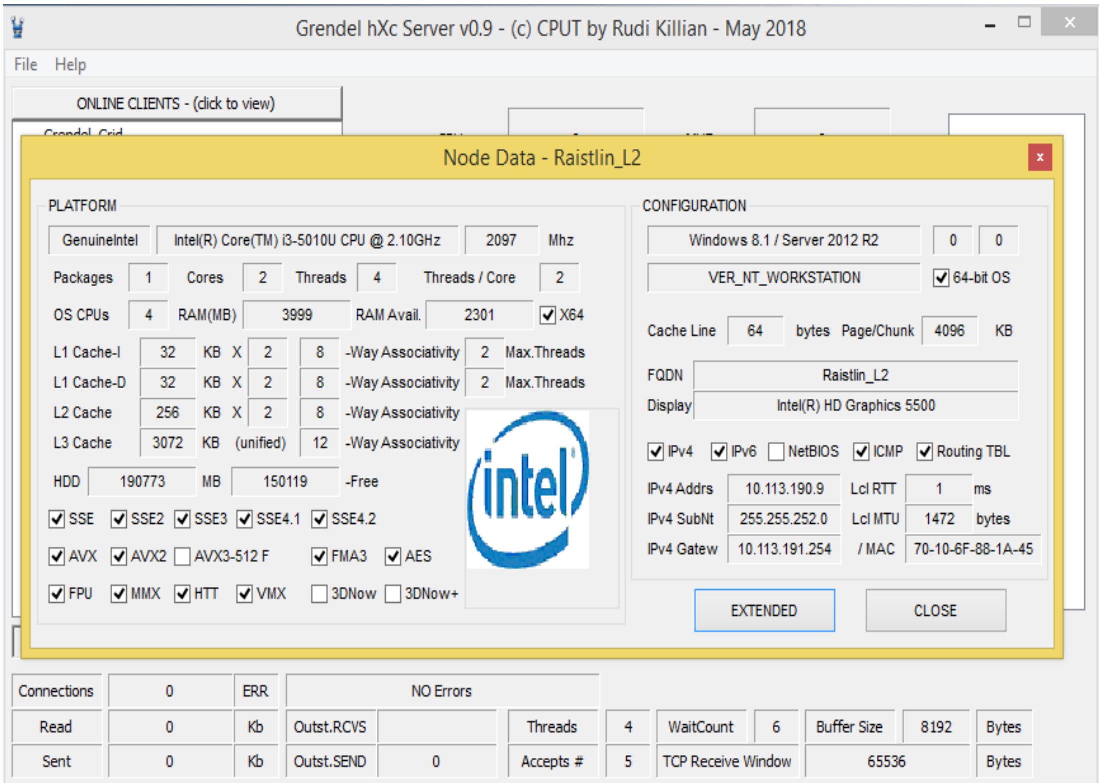
Data size: 1020 bytes

C3 Screenshot Output of Client Module



```
-----------------------------------------------------------------------
                 Grendel hXc Client Started
-----------------------------------------------------------------------
(c) Rudi Killian, Cape Peninsula University of Technology, MASM SDK
-----------------------------------------------------------------------
Loading....System Profiler
Completed profiling in 23 milliseconds
No serious platform errors detected
Spooled-up IOCP client services
Connecting to Grendel hXc Server on 192.168.1.101
----------------------------------------------------------------------
-1 10060 -Error: Server Connect Failed
Press any key to continue ...
```

C4 Screenshot Output of Server Module


Grendel hXc Server v0.9 - (c) CPUT by Rudi Killian - May 2018
File
Help
ONLINE CLIENTS - (click to view)
Node Data - Raistlin_L2
x
EXTENDED
CLOSE

PLATFORM

GenuineIntel | Intel(R) Core(TM) i3-5010U CPU @ 2.10GHz | 2097 Mhz

Packages 1 | Cores 2 | Threads 4 | Threads / Core 2

OS CPUs 4 | RAM(MB) 3999 | RAM Avail. 2301 | ☑ X64

L1 Cache-I	32	KB X	2	8	-Way Associativity	2	Max.Threads
L1 Cache-D	32	KB X	2	8	-Way Associativity	2	Max.Threads
L2 Cache	256	KB X	2	8	-Way Associativity		
L3 Cache	3072	KB (unified)		12	-Way Associativity		

HDD 190773 MB | 150119 -Free

☑ SSE ☑ SSE2 ☑ SSE3 ☑ SSE4.1 ☑ SSE4.2

☑ AVX ☑ AVX2 □ AVX3-512 F ☑ FMA3 ☑ AES

☑ FPU ☑ MMX ☑ HTT ☑ VMX □ 3DNow □ 3DNow+

CONFIGURATION

Windows 8.1 / Server 2012 R2 | 0 | 0

VER_NT_WORKSTATION | ☑ 64-bit OS

Cache Line 64 bytes | Page/Chunk 4096 KB

FQDN Raistlin_L2

Display Intel(R) HD Graphics 5500

☑ IPv4 ☑ IPv6 □ NetBIOS ☑ ICMP ☑ Routing TBL

IPv4 Addrs	10.113.190.9	Lcl RTT	1	ms
IPv4 SubNt	255.255.252.0	Lcl MTU	1472	bytes
IPv4 Gatew	10.113.191.254	/ MAC	70-10-6F-88-1A-45	

Connections	0	ERR	NO Errors							
Read	0	Kb	Outst.RCVS		Threads	4	WaitCount	6	Buffer Size	8192	Bytes
Sent	0	Kb	Outst.SEND	0	Accepts #	5	TCP Receive Window		65536	Bytes