FINITE ELEMENT ANALYSIS IN A CLOUD COMPUTING ENVIRONMENT

A Dissertation

by

Nitel Muhtaroğlu

Submitted to the
Graduate School of Engineering and Science
In Partial Fulfillment of the Requirements for
the Degree of

Doctor of Philosophy

in the
Department of Computer Science

Özyeğin University
January 2019

FINITE ELEMENT ANALYSIS IN A CLOUD COMPUTING ENVIRONMENT

Approved by:

Asst. Prof. Dr. İsmail Arı, Advisor
Department of Computer Science
Özyeğin University

Asst. Prof. Dr. Didem Unat
Department of Computer Engineering
Koç University

Assoc. Prof. Dr. D. Turgay Altılar
Department of Computer Engineering
Istanbul Technical University

Assoc. Prof. Dr. G. Güven Yapıcı
Department of Mechanical Engineering
Özyeğin University

Date Approved: 4 January 2019

Asst. Prof. Dr. T. Barış Aktemur
Department of Computer Science
Özyeğin University

To my daughter, Aysu.

ABSTRACT

In this thesis, the challenges faced and lessons learned while establishing a large-scale high performance cloud computing service that enables online mechanical structural analysis and many other scientific applications using the finite element analysis (FEA) technique, will be described. Within an High Performance Computing (HPC) environment, several jobs with different demands can co-exist thus it becomes a challenge for the service provider to efficiently utilize its own resources while also satisfying the quality expectations of job submitters. Such a service is intended to process many independent and loosely-dependent tasks concurrently. In order to reach optimal job scheduling metrics each job type that can be submitted to the cluster must be carefully examined, its space and time characteristics must be well-understood and quantified. Challenges faced include accurate characterization of complex FEA jobs, handling of many-task mixed jobs, sensitivity of task execution to multi-threading parameters, effective multi-core scheduling within a single computing node, and achieving seamless scaling across multiple nodes. It is found that significant performance gains in terms of both job completion latency and throughput are possible via dynamic or "smart" batch partitioning and resource-aware scheduling compared to the naive Shortest Job First (SCF) and aggressively-parallel scheduling techniques.

Chapter 3 of this thesis present an end-to-end discussion on the technical issues related to the design and implementation of a new cloud computing service for finite element analysis (FEA). Several design choices for HPC services at different layers of the cloud computing architecture are investigated to simplify and broaden its use cases. Investigations start with the software-as-a-service (SaaS) layer and compare parallel linear equation solvers. In order to minimize job latency and maximize the overall job throughput, several matrix characteristics are perceived. Developing such an understanding is also crucial for HPCaaS systems to automatically select the amount of computing resources per job. In following sections, the design of a "smart" scheduler that can dynamically select some of the required parameters, partition the workload and schedule it in a resource-aware manner will be demonstrated. Results showing that an up to 7.53x performance improvement over an aggressive scheduler using mixed FEA loads, will be presented. In addition to the performance studies, a complementary discussion on critical issues related to the data privacy, security, accounting, and portability of the cloud service will also be given.

The new trend in engineering is to solve complex computational problems in the cloud over HPC services provided by different vendors. To further deepen the analyses of workloads representing HPC-related tasks in science and engineering, in chapter 4, performances of direct vs. iterative linear equation solvers are compared to help with the development of job schedulers that can automatically choose the best solver type and tune them (e.g. precondition the matrices) according to job characteristics and workload conditions that are frequently encountered on HPC cloud services. As a proof of concept, three classical elasticity problems will be used, namely a Cantilever beam, Lame problem and Stress Concentration Factor (SCF). These models theoretically represent many real-life mechanical situations in structural engineering, namely aerospace, automotive, construction and machinery industries. The representative linear problems are meshed with increasing granularities, which leads to various matrix sizes; largest having 1 billion non-zero elements. Detailed finite element analyses over an IBM HPC cluster are executed. First, a multi-frontal parallel is used, sparse direct solver and evaluate its performance with Cholesky and LU decompositions of the generated matrices with respect to memory usage, and multi-core, multi-node execution performances. As for the iterative solver, the PETSc library is used and carried out computations with several Krylov subspace methods (CG, BiCG, GMRES) and preconditioner combinations (BJacobi, SOR, ASM, None). Later in Chapter 4, the direct and iterative solver results are compared and contrasted in order to find the most suitable algorithm for varying cases obtained from numerical modeling of these three-dimensional linear elasticity problems.

In addition to aforementioned studies, as a supplementary research, infrastructure-as-a-service (IaaS) layer for HPC is examined and characteristics like application performance, load isolation, and deployment speed issues using application containers (Docker) are observed. These characteristics are also compared to physical and virtual machines (VM) over a public cloud. For this purpose, HPC-specific deployment using application containers technology is evaluated and performance metrics are examined in order to contribute to evaluation of these technologies for job schedulers to be used on Cloud Computing infrastructures. This phase of the research focuses on the understanding the behavior of cloud computing infrastructures under circumstances where deployment and utilization of containers (Docker) with a chosen software is necessary.

To summarize, this multi-disciplinary doctoral thesis covers most of the critical aspects and computational challenges of providing FEA in the cloud for structural mechanics including ease of deployment, batch-level performance, job-level isolation, financial accounting and content security. It utilizes several modern software tools and techniques, while also contributing new ones to the literature.

ÖZETÇE

Bu tezde, bulut bilişim ortamında çalışan büyük ölçekli bir yüksek başarımlı hesaplama düzeneği kurulumu sırasında karşılaşılan güçlükler ve öğrenimler aktarılacaktır. Özel olarak çevrimiçi bir mekanik yapısal analiz sistemine, genel olarak ise bir çok bilim ve mühendislik alanına uygulanabilecek bu öğrenimler, sonlu elemanlar yöntemine özel bir vurgu yapılarak incelenecektir.

Yüksek başarımlı hesaplama ortamlarında aynı anda birden çok farklı türde ve özellikte iş aynı anda bulunurlar. Hizmet sağlayıcının iş gönderen kullanıcıların beklentilerini karşılamak ve aynı anda da yüksek başarım ortamının en verimli şekilde kullanılması gibi birbiriyle çelişen iki kaygıyı dengelemesi gerekmektedir. Böyle bir düzeneğin verimli çalışması için ise küme üzerindeki işlere ve kümenin verili andaki durumuna göre davranabilen akıllı bir zamanlama yönteminin geliştirilmesi kritik önem taşımaktadır. Bu zamanlama yönteminin geliştirilmesi için gerekli ön çalışmalar, kümenin çözümlemeye çalıştığı iş türlerinin anlaşılıp sınıflandırılması, bu yüklerin zaman ve alan ihtiyaçlarının anlaşılması ve nicel olarak tanımlanmasını içermektedir. İkinci aşamada ise, bu iş yüklerinin çok çekirdekli ve çok düğümlü ortamlarda kaynakların verimli kullanılmasının sağlacak şekilde nasıl zamanlanabileceği sorusunun cevabının araştırılması gerekmektedir. Araştırmalar sırasında iş türlerinden ve hesaplama kümesinin gerçek zamanlı durumundan haberdar olan akıllı bir zamanlama yöntemi kullanılarak, kullanıcılar açısından gecikmeleri azaltmanın, hizmet sağlayıcılar açısından ise verimliliği arttırmanın mümkün olduğunu, bu tür durumlarda yaygın olarak kullanılan en kısa olanın önce çözümlenmesi veya agresif olarak bütün kaynakların kullanılması yöntemlerinden daha iyi sonuçlar alınabildiğini gözlemlenmiştir.

Bu tezin ilk bölümleri sonlu elemanlar yöntemini bir bulut bilişim hizmeti olarak

sunmak için gereken tasarımların ve uygulamarın teknik bir tartışmasını içermektedir. Bu teknik tartışma özünde, kurgulanan yüksek başarımlı hesaplama hizmetinin değişik katmanlarda incelenmesi ve mimarilerin oluşturulmasını kapsamaktadır. İlk olarak yazılım hizmeti katmanına odaklanarak doğrusal denklem takımlarının çözümlerini incelenecektir. Ardından bu incelemeler sırasında hesaplamalara temel oluşturan matrislerin karakteristiklerinin işler kümeye gönderilmeden belirlenmesi ve iş hesaplama değişkenlerinin bunlara göre güncellenmesinin verimliliğinin arttırılmasına ve aynı zamanda gecikmelerin azaltılmasına çok büyük katkıları olduğunu gözlemlediği bulgular paylaşılacaktır. Bu giriş kısmını takip eden diğer bölümlerde ise tasarlanan akıllı zamanlayıcı ve başarıma olan 7.53x hızlandırıcı etkisi deney sonuçları ile beraber gösterilecektir. Başarım üzerine verilecek örneklerin ardından ise veri güvenliği, fiyatlandırma ve taşınabilirlik gibi konular da incelenecektir.

İlerleyen bölümlerde ise mühendislik hesaplamalarında yeni yeşermekte olan bir yaklaşım olan yüksek başarımlı bulut bilişim hizmetlerinin kullanılmasının daha etkin hale getirilmesine katkı vermek amacıyla, direkt ve iteratif doğrusal denklem takımı çözücülerinin incelenmesini çeşitlendirilecektir. Ayrıca akıllı zamanlayıcının sadece donanım anlamında hesaplama parametrelerini değil bunun yanı sıra da çözümlenecek işin yapısına göre yazılım parametrelerine de müdahale etmesini sağlamak amacıyla gerçekleştirilen araştırmaların sonuçlarına yer verilecektir. Bu araştırmalar sırasında kullanılan gerçek hayattan alınma doğrusal elastisite problemleri hakkında da kısa bilgilere yer verilecektir. Çeşitli çözünürlüklerde ayrıklaştırılarak örgüleri oluşturulmuş bu modellerin kullanıldığı, geliştirilen akıllı zamanlayıcının değişkenlerini belirlendiği, Cholesky, LU gibi direkt çözücülerin yanı sıra çeşitli Krylov Altuzay Yöntemleri ile de sınandığı deneylerin sonuçlarını paylaşılacaktır. Bu bellek kullanımı, çoklu çekirdek, çoklu düğüm koşum davranışlarını daha sonra lineer elastisite problemlerinin çözümü için gerekli donanım ve yazılım değişkenlerini verimli şekilde ayarlamak için temel olarak kullanabilecektir.

Bahsedilen çalışmalara ek olarak ise altyapı servisi katmanına da odaklanıp, yüksek başarımlı hesaplama için uygulama kapları kullanıldığında başarım, yalıtım ve kurulum hızı gibi parametrelerin davranışlarını gözlemlendiği çalışmalara da yer verilecektir. Uygulama kaplarının, fiziksel ve sanal makinalarla davranış farklarının da yorumlanacağı bu kısım tezin son bölümünü oluşturacaktır.

ACKNOWLEDGEMENTS

Firstly, I would like to express my sincere gratitude to my advisor Professor İsmail Arı for the continuous support of my Ph.D study and related research, for his patience, motivation, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis. Without his guidance and mentoring submission of dissertation would not be possible.

Besides my advisor, I would like to thank the rest of my thesis committee: Professor Barış Aktemur, Professor G. Güven Yapıcı, for their insightful comments and encouragement, but also for the hard questions which incented me to widen my research from various perspectives.

I am grateful to my family, who have provided me through moral and emotional support in my life. I am also grateful to my wife, Nilay for her support, trust, understanding and friendship along the way.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

According to the U.S. National Institute of Standards and Technology (NIST) [1], [2]:"Cloud Computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources ... that can be rapidly provisioned and released with minimal management effort." NIST further differentiates cloud as having "five essential characteristics, three service models, and four deployment models". Cloud services should essentially have on-demand network-based accessibility, resource pooling and rapid elasticity characteristics, could be provided via software, platform or infrastructure as-a-service models (as illustrated in Figure 1), and be made available through private, community, public or hybrid deployments. An infrastructure service (or IaaS) virtualizes the capacities of physical computing hardware such as the CPU, storage or networking equipment and provides remote, shared access to these virtualized resources. Platform services (or PaaS) are usually exposed via web services and are shared among different desktop applications as well as online software services. End-user software services (or SaaS) hide the infrastructure or platform specific details from the clients and they are usually accessed via web portals. Each layer can be provided on top the other (e.g. a platform service can be deployed in virtual machines hosted by an IaaS provider), but many SaaS or PaaS providers still prefer to provide services on top of their own infrastructure today. Different service providers operating at the same layer are beginning to standardize their interfaces to enable "horizontal integration" (e.g. open virtual machine formats). However, "vertical integration" among different cloud service layers and providers is still an ongoing research area. The results of these investigations will affect large-scale governmental and business cloud deployment decisions.

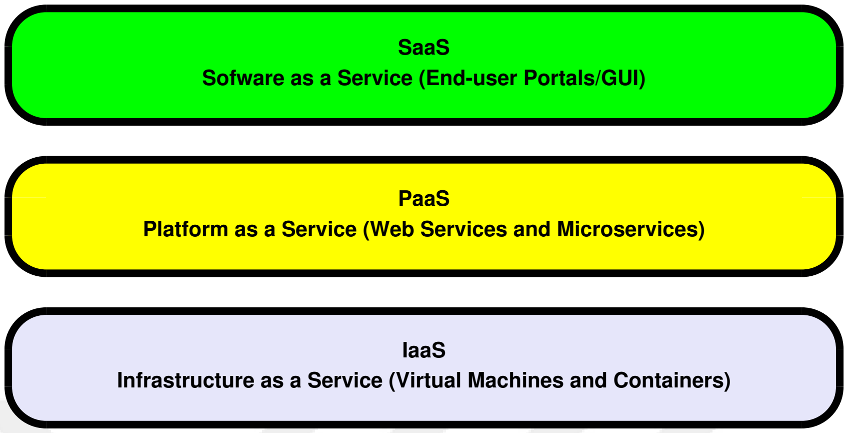

SaaS
Sofware as a Service (End-user Portals/GUI)
PaaS
Platform as a Service (Web Services and Microservices)
IaaS
Infrastructure as a Service (Virtual Machines and Containers)

Figure 1: Different service models for cloud computing and the logical layering among them.

Experiences with the engineering and scientific communities reveal the need for cloud computing services that can be shared among different disciplines for solving common problems. The current practice for solving large-scale high-performance computing (HPC) problems is to acquire expensive hardware resources and gain special Information Technology (IT) skills to manage those [3]. While IT management is not the main goal of the engineering community, ultimately significant time and effort is spent on installing, maintaining, and tuning computing resources. Furthermore, most hardware resources and associated software licenses remain underutilized after a few initial runs. People who do not have the skills, time or finances to take on these IT challenges are deterred from pursuing this path.

Unfortunately, most of the HPC cloud services offered today are for advanced users only, who can understand both parallel computing paradigms and IT management including virtualization issues. Some acclaimed HPC services just provide a set of physical or virtual machines to their end users, i.e. Infrastructure as a Service (IaaS). Others provide HPC as a platform service (PaaS), which in addition to infrastructure, establish clusters bound by communication mechanisms such as Message Passing Interface (MPI) [4] and queues for job management. These two models make a good attempt in addressing the low-utilization problems witnessed in privately-held supercomputers, but do not address the lack of highly-skilled experts both in advanced computational methods and cluster management, simultaneously. For most engineers who are only experts in their respective fields, HPC Software as a Service (SaaS) model is the only viable cloud model. In HPC-SaaS model, sector-specific software packages with interactive graphical user interfaces are provided over the Internet [5]. But unfortunately, none of these services have automated and advanced mechanisms for selecting the best solver and system settings for submitted jobs [6].

Cloud Computing models offer tremendous cost savings and sharing opportunities to technical communities, (especially those in developing countries) that deal with similar engineering problems including finite element analysis (FEA) [7]. Within this scope, cloud computing paradigm has been well-accepted, since gives equal opportunity to engineers, researchers, students as well as small-medium enterprises in accessing flexible and scalable computational resources anytime and anywhere in the world, and as long as they need it. There is no up-front capital investment for servers and the operational expenses are manageable. The premise is that everyone can now attempt to solve large and complex engineering problems.

Co-operation among distributed and dynamic engineering teams is another motivating scenario for the use of High Performance Computing-as-a-Service (HPCaaS). Large-scale engineering simulations (e.g. for aerospace, automotive, and civil engineering) and big data jobs can still consume days stealing from design time, thus business opportunities for enterprises [8]. Integrating development, testing and deployment operations (DevOps) capabilities with application containers such as Docker into HPC can be the remedy. Democratizing HPCaaS requires automation both at the cloud platform and infrastructure layers, which will be explored in this thesis.

Recent developments in Computer-Aided Design (CAD) software packages and Additive Manufacturing technologies enable engineers, researches and employees of small-medium sized companies, to rapidly develop complex 3D structures [9]. However, in such situations, there is also a need to understand the mechanical behavior of such components to confirm their safe and satisfactory performance on the field [10]. A significant portion of such commonly-encountered designs turn into demanding computations that require resources beyond the capacity of desktop computers. High Performance Computing (HPC) cloud services [11], [12], [13], a.k.a HPCaaS, offers a significant alternative by letting everyone to utilize scalable computational resources on demand. Removing the need for up-front capital investment for servers and setting the the usage fees help people to access this once luxurious field. However, in engineering and science, complicated physical problems are defined as partial differential equations (PDE), and numerical solutions to such problems often demand extraordinary amounts of resources to return a meaningful insight. Also if the multi-tenant scenarios that are frequently encountered in cloud computing services are taken into account, sustaining an adequate quality of service becomes demanding.

Nowadays, one of the main questions about HPC acquisition is whether "to buy or rent" these HPC resources. Alternatives include: Physical HPC, Virtual HPC, and recently added Dockerized HPC models. Acquiring a physical HPC cluster requires both a Capital Expense (CapEX) for buying the IT equipment and Operational Expenses (OpEx) afterwards, which cannot be justified for tens of years if the cluster is not highly utilized (e.g. $< \%20$). The Virtual HPC model, a.k.a. HPC-as-a-Service or the public Cloud HPC, removes the CapEx and minimizes the OpEx costs. Utility costs that include electricity, CRAC cooling, backup storage, security, and availability are all merged into a single rate such as \$0.1/CPU-hour and \$0.02/GB-month [14]. While the accounting can get quite complex due to variety of server capacities, data center regions, and market models, one can easily reserve a server (e.g. Amazon EC2 m4.large instance) for \$500/year [14], which includes all utility costs and guarantees business continuity. Therefore, HPCaaS has become a price-competitive, flexible, and powerful proposition [15]. The reasons for relatively slower adoption of this model within the HPC community has been due to security concerns, fear of vendor lock-in, and lack of understanding of the HPC technical issues among the cloud service providers (e.g. lack of the needed cluster-level control, varying performance, network contentions, and cluster availability when instances are reserved in larger counts). Yet, the maturity, stability and multi-vendor availability of HPC cloud services are also increasing over time.

In addition to the mentioned infrastructure related issues, in HPC software environments, there is an ever increasing task of maintaining software packages, handling dependent libraries, and making application-specific configurations [16]. There are still significant performance overheads for doing these tasks in the Virtual HPC model. Copying virtual machines (VMs) inside or among public-private clouds, installing special operating systems (OS) and HPC software can be a hassle and consume time. There can also be higher OpEx costs associated for storing many pre-configured VMs for different HPC applications in the long-run. Fortunately, this cost can now be reduced with the container technology.

From the HPCaaS providers' point of view, such a service could only be sustainable and profitable, (1) if significant number of customers can be served by the very same cluster, and (2) the contentment of end-users can be achieved via low-pricing, high performance, and predictability. Since satisfying both the maximum job throughput requirements of service providers and the minimum latency and predictability requirements of the customers is a contradictory situation, it is still possible to find an equilibrium point that satisfies both sides through solver-aware and task-aware scheduling.

A large set of HPCaaS demanding problems encountered in industry are related to physics based problems that are especially originating from solid and fluid mechanics. Such problems eventually map to solving the partial differential equations (PDE) based on the finite element method (FEM). FEM is a generally-applicable numerical method to approximately solve partial differential equations (PDE) and for most of the serious industrial applications requires HPC setups. Figure 2 shows some of the application areas of FEA including mechanical structural analysis, heat transfers, fluid dynamics, acoustics, and electromagnetic modeling. Several other related numerical methods have been developed in the past (FEM, FDM, FVM, BEM in Figure 2), each of which may be more suitable for different application areas due to special characteristics of that given problem space. In addition, numerous open-source and proprietary software tools that enable application of these numerical methods to real-life problems are available in the market as desktop or mainframe applications. Some of the well-known packages include Nastran, Ansys, Abaqus, CalculiX, Code Aster, and various others [17]. However, the installation and large-scale maintenance of these tools over continuously evolving operating system (OS), processor and cluster technologies can be costly and cumbersome for the end users. Therefore, to lower the barrier of entry for technical individuals as well as small and medium businesses (SMB) [18], FEA applications can be deployed as a cloud computing services. In order to use such a domain specific SaaS, all that the users will need is a personal computing device with a browser and an Internet connection to enable them to access the HPC cloud service for FEA.

Other components shown Figure 2 will be described in detail later in Section 3.1.

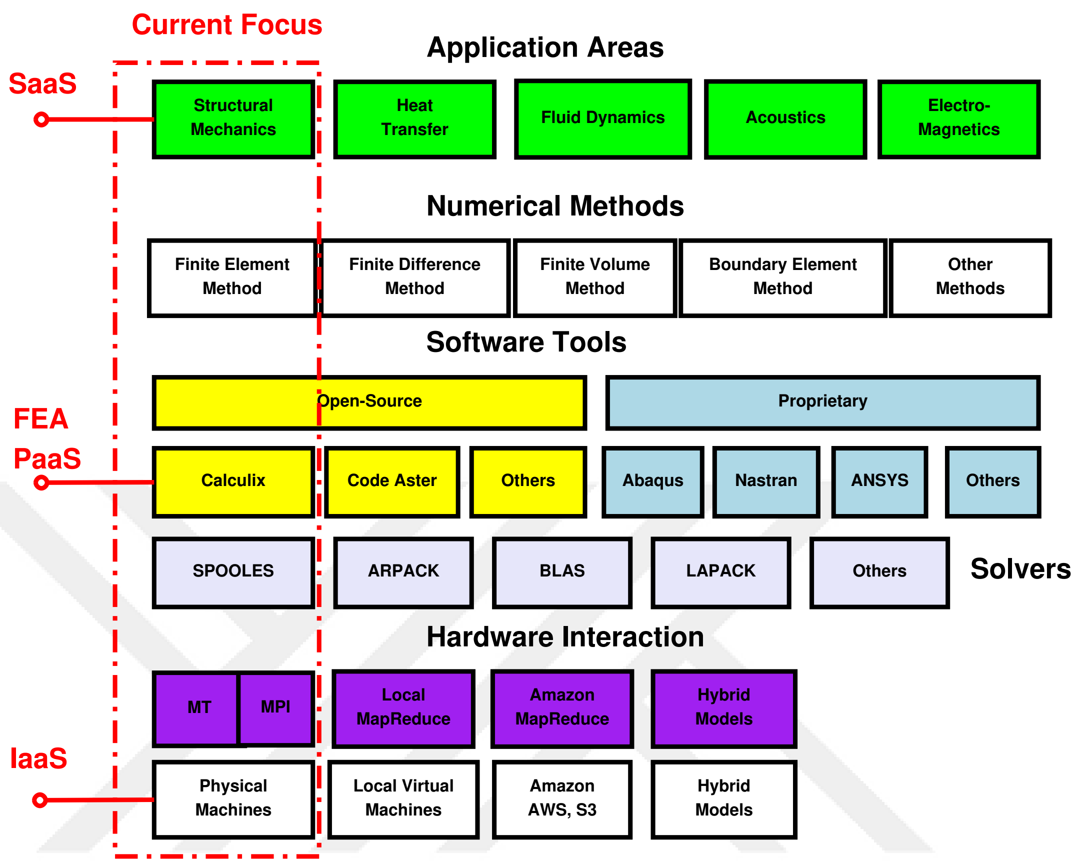


Figure 2: Layers and logical components of a modern HPC cloud service are shown. Current research focus is on developing the FEA PaaS.

Today it is common, even for a moderate engineering analyses used in industry to have millions of degrees-of-freedom. In addition to this fine meshing practice, industry standards enforce engineers to incorporate non-linearities such as contact modeling, large deformations, and complex materials into their analysis. The time for such simulations can be measured in hours and even days. Such computationally demanding activities are usually carried out by multiple engineers simultaneously, especially during peak periods close to project deadlines. All these considerations make it critical for research and development (R&D) institutions to provision high-performance computing (HPC) resources, plan job scheduling, and do smart resource allocations.

In order to determine fundamental constraints that drive the performance of such an online service and to design a scheduling algorithm, the computational characteristics (RAM and multi-core CPU usage, I/O behavior) of illustrative mechanical tasks are analyzed and identified, a scheduler concept that could smartly and simultaneously configure optimal resource allocations per task and per mixed-batches of tasks is proposed. Remarkable performance increases over a conventional (First-In-Fist-Out) and an aggressive (all-core parallelized) schedulers are observed. However, only SPOOLES [19] was tested as the linear equation solver in the research that forms the body of Chapter 3, as there were numerous other tunable parameters to deal with. In Chapter 4, performances of direct vs. iterative linear equation solvers and preconditioners with representative mechanical workloads will also be compared. For this purpose, as a first step, direct solver characteristics using Cholesky and LU decompositions of the matrices and report the memory usage and parallel execution times are tested. In order to test the iterative solver characteristics, benchmarks with several Krylov subspace methods (GMRES, CG, BiCG) and preconditioner combinations (ASM, SOR, BJacobi, None) are executed. Previously [19], solely the NZE properties of matrices were examined, whereas in this thesis also the corresponding condition numbers (before and after preconditioning) of resulting matrices are taken into account.

To sustain high-performance in a FEA service, first of all, there is a need to accurately characterize workloads. As FEA is a broad area of research, in this thesis, the focus is set mainly on mechanical structural analysis, which is used ubiquitously in automotive, aviation, home appliances, construction and defense industries as well as academia. The lessons learned and methods developed will be generally applicable to other FEA and HPC subject areas, since the underlying mathematical and computational principles are mostly similar. In following chapters, structural mechanics benchmarks using open source software tools and over local physical machines will be executed. Additionally, later in the thesis, a discussion on extension of the work into other SaaS application areas and use virtual or remote infrastructure-as-a-service (IaaS) resources [20] within the service has been made. Contributions of this thesis are as follows:

- Design and implementation of a new online FEA cloud service different from existing offerings. This service provides shared services at the software-level (SaaS-PaaS) whereas most existing services are based on hardware sharing (IaaS).
- Performance characterization of representative FEA workloads (based on real-life objects like beams, rotors etc.) and their mixes over shared memory (multi-core) and distributed memory (multi-node) resources.
- A comprehensive evaluation of alternative task execution, scheduling strategies and showing performance improvements using smart scheduling.
- Discussions about critical underlying Linux OS process and memory management mechanisms that most other FEA works stay oblivious to.
- A complementary discussion on cloud service privacy, security, accounting, and portability issues, the lack of which can lead to breaking or abandonment of this service by clients.
- Evaluation of application containers within the HPCaaS context.
- A "smart scheduler" prototype design

The rest of the thesis is organized as follows. Section 2 describes the previous work including issues related to cloud security and privacy. This section also presents a literature review on similar research and technologies. Section 3 overviews the design of the FEA service architecture, describes the benchmarks used and presents results for workload characterization. This section also describes the experimental setup and gives detailed performance results. Section 4 demonstrates the different combinations of Direct and Iterative Solvers with several preconditioners and also discusses the utilization of application containers and their evaluations using several types of HPC workloads. Section 5 discusses some of the future work and concludes the thesis.

CHAPTER II

PREVIOUS WORK AND LITERATURE REVIEW

There are hundreds of supercomputers in the world [21] and some of these may be providing FEA computation services along with other HPC services. However, these supercomputers are usually used by selected research communities and are not available to the broader public. During the last few years, in order to broaden the utilization of these resources, there have been attempts to establish HPC cloud services or HPC as a service. In February 2010, SGI announced Cyclone–HPC Cloud [22]. Cyclone implements both SaaS and IaaS models. Cyclone's SaaS model has commercial and open-source software solutions for several disciplines including computational biology and chemistry, fluid dynamics, FEA, and electromagnetic. In their SaaS model, the licensing issues for the commercial software are left to the clients. Cyclone's IaaS model lets the clients install and run their software on SGI's hardware. Penguin Computing also provides an HPC service called Penguin Computing on Demand (POD). POD serves at the IaaS level and makes CPU, GPU, and storage resources available through the Internet for technical computing. It does not provide PaaS or SaaS models.

In 2007, Sun Microsystems (now Oracle) claimed to have made some of the open-source Computer Aided Engineering (CAE) packages including CalculiX available through the Sun Grid [23] . In this system, the user had to prepare the input file locally with a pre-processor and upload it to the Sun Grid. Next, CalculiX solver was used to solve the problem on Sun's infrastructure service at 1$/CPU-hour. The service is currently not available. Recent research also suggests that IaaS-based HPC cloud services [24] [25] can suffer in performance especially due to VM resource competition and lack of low-latency interconnection needed by specialized parallel engineering simulations.

FIDESYS Online (http://online.cae-fidesys.com/) is another FEA cloud service at the alpha stage (as of 2012) that evolved from a packaged program [26]. Similar to the proposed Cloud FEA service, it allows application of meshing, boundaries and forces to uploaded CAD files followed by the calculations and analysis in the cloud. They do not focus on job characterization for effective multi-core or multi-node scheduling as done in this thesis. FEMHub, hp-FEM Group, University of Nevada, Reno (http://femhub.org/) is another FEA cloud service project in progress, which provides software-layer sharing similar to ours. Users can write Python code through the web interface and access FEMHub's FEA engine called Hermes. This service also does not focus on job characterization and scheduling issues.

Scheduling, especially production scheduling, is also a major research field inside Industrial Engineering. Chiang et al. [27] address the Flexible Manufacturing Scheduling (FMS) problem with Critical Ratio-Based heuristics and genetic algorithm. Other algorithms called dispatching rules [28] include FIFO, Shortest Processing Time (SPT) [29], Critical Ratio (CR), EDD (Earliest Due Date), and Shortest Remaining Processing Time (SRPT). Park et al. [29] proposed a multi-class SVM-based task scheduler for heterogeneous grids that maps queued tasks to machines based on their sizes and machines' loads as well as their machine computing powers to minimize the total completion time (makespan) of all tasks. Their results suggest that SVM scheduler can closely follow the performance of the best performing heuristics (Early First, Light Least) and soundly outperform Round Robin scheduler. Breslow et al. [30] develop a technique called job striping that colocates job pairs on the same node in order to increase throughput and energy-efficiency. This approach can increase throughput up to %26 and energy efficiency up to 22 % in environments where heterogeneous workloads are computed. Work in this thesis differs from [29] and other prior scheduling work such that FEA tasks can be split further and a decision MP vs. MPI parallelization models can be made before scheduling tasks onto machines. Therefore, there is neither a bound nor a dependence on the performance of other heuristics. In fact, in Table 5, outperforming of both SJF and Aggressive schedulers is shown. Doulamis et al. [31] study task allocation in Grids and suggest an earliest completion time scheduling algorithm based on estimated task completion times. However estimating completion times for nonlinear tasks is a challenging research problem. Belytschko and Mish [32] examine the computability of nonlinear problems in solid and structural mechanics problems. A measure of the level of difficulty ($L : 1 \rightarrow 10$) is proposed and examples of typical engineering simulations are classified by this measure. In the future, the smart scheduler may be compared with deadline-based algorithms from different research fields.

For FEA of assembled complex structures finding and tracking the dependencies among tasks and their execution ordering is also an interesting research area. Amoura et al. [33] discuss the issue of task preemption for independent tasks over multi-processor to obtain minimum makespan. Calculating the number of different scheduling alternatives given a batch of independent tasks and computing resources is a known "counting problem" whose solution is given by the Stirling number of the second kind S(n, j), where n denotes the number of tasks and j the number of cores. The research model for hierarchical multiprocessor tasks (M-tasks) [34] also creates a data and control dependency graph among parallel tasks and schedules tasks to multi-core resources by layers (root-to-leaves) to assure overall progress. In this thesis, analysis of assembled (multi-part) systems and computational dependencies among them are taken into consideration.

Cloud-based mechanical structural analysis efforts have been initiated by a previous research [35] and additional contribution in this direction [19] were made. Related research and development efforts include SimScale [5] and similar others [36]. SimScale provides its customers an interactive and web-based finite element cloud service for mechanical, thermal and CFD analysis using OpenFOAM[37], CalculiX [38] and CodeAster [39] FEA engines at the back-end. While they provide suggestions about some "rules of thumb" [11] for selecting direct vs. iterative (PETSc) solvers, the process is currently not automated. They suggest using direct solvers for job sizes smaller than 50,000 degrees of freedom (DOF). This thesis aims to fill the gap in practice by designing automatic procedures for job characterization and assignment of necessary resources.

A study on performance of parallel linear equation solvers has been carried out by Bui and Meschke [40], They found that system re-ordering and scaling perform effectively and enhance the stability of the outcome. They have executed their preconditioning collations to develop domain-specific (tunnel engineering) knowledge. In another related study, Pommerell and Fichtner [41] have also examined the behavior of iterative solvers on VLSI device simulations. Sourbier et al. [42] developed a hybrid iterative/direct solver combination in order to model the frequency-domain seismic waves. They are still working towards generalizing their perspective for other fields. Again, these studies are complementary to this research that focuses on the structural analysis domain.

Increased availability and affordability of 3D printers and improved additive manufacturing techniques for various materials motivates the use of topology optimization techniques [43], since these techniques save immense time during design and production [12]. Topology optimization starts with a virtual block of material and it automatically reduces mass up to a desired volume fraction (e.g. $0.2x$) by removing the unnecessary parts without compromising the desired stress-strain constraints of the object under given forces. Recently, this technique has been made available as a cloud service [13], so that designers can quickly reach the optimal concept design that meets their requirements without hardware investments. This work is complementary to these cloud services, as it aims to organize and speed up the solution process.

Process Containers were first implemented and integrated into the Linux Kernel in 2007 starting with version 2.6.x by Paul Menage, et al. [44]. The goal was to provide resource isolation (CPU, memory, I/O, network) and prioritization among controlled group (cgroups) of processes. Light-weight containers can be setup, configured, shipped, copied, deployed, and terminated much faster than Virtual HPC, since they run on the same OS kernel share this kernel's resources and libraries. They are also designed to isolate jobs within a cluster; but this is a claim to be tested extensively. This isolation within the cluster should make different HPC applications co-exist and utilize the underlying hardware resources without any conflicts. Finally, containers can remove the need for queue scheduling and management software such as PBS [45] and SLURM [46], since there is a separate and isolated process for every HPC job to be executed.

Container technologies can be classified into two as "application containers" and "system containers" [47]. Docker is an example for the former, whereas Linux Containers (LXC) [48] and OpenVZ for the latter. App containers execute a single process, whereas system containers have full OS stack. Due to its light-weight approach and ease of deployment, Docker [49] is the most widely adopted technology among the systems community for now. It is available for Linux, Windows, and Unix-based OS such as MacOS. Scientists can easily share pre-packaged images that are loaded with scientific software and data via Docker Hub. Images can be layered (using a layered file system) to share functionality and save disk space. However, sharing can create resource contention and reduce performance if not handled properly.

Ruan et al. [47] investigate container performance in the cloud and find that extra VM layer adds $\sim 5\%$ overhead in CPU-intensive SPEC benchmarks and almost no overhead in STREAM memory benchmark. They also found that AUFS layered-FS can create $\sim 40\%$ additional latency in read/write operations, due to file copies among layers for write operations. But, this problem was recently solved by Shifter [50] via write caching. Shifter aims to accelerate science via "container computing" and REST-based web interfaces for image management. It also claims startup times that are $2x - 5x$ faster due to their per-node writable cache that especially benefits Python HPC applications that scan file system paths to find dynamically linked library (DLL) files.

Recently, there has been an emerging interest in the use of containers for managing scientific workloads [49], HPC [48] and in the cloud [51]. Skyport utilizes Docker containers to ease software deployment for scientific workflows [52],[53]. Babu finds LXC to give better result in packet transfers [54] and to have less fluctuations where processes communicate with each other frequently. Xavier, et al. [48], [55] measure the performance of containers in HPC environments. They conclude that containers could obtain a very low overhead leading to near-native performance, but performance isolation in containers is immature. However, these studies do not discuss the distinctions between different containers and the impact of adding an extra virtualization layer between the bare metal and containers.

Within the public cloud, Amazon EC2 Container Services (ECS) and Google Container Engine (GKE) both primarily support Docker containers. GKE is built upon the open-source Kubernetes system for container deployment and orchestration. Ruan et al [47] find up to 11% difference between ECS and GKE using Hadoop, Spark, Kmeans benchmarks, but they also admit that they cannot fully control the VM instances (IaaS) underneath their containers and it may not be fair to make clear conclusions. In this research, bare metal physical servers in the cloud are also used for control purposes. Microsoft also supports containers via public Azure Container

Service. However, it is quite debatable whether it is a good practice to run containers inside virtual machines (VMs), since a major reason for their invention was to replace heavy-weight VMs. Containers are also beginning to finding sector-specific applications. For example, in the mobile telecommunications world, a Docker container can be used to contain a small cell service [56] inside a base station and small cell identifiers can be mapped to docker ports.

Altiscale provides Hadoop-as-a-Service, where clients load their data into quickly bootable Hadoop clusters (Hbase, Hive included). It had to solve the multi-tenancy problem reliably, at scale and had to optimize usage of CPU and storage resources. Alternatives were running Docker over Hadoop and benefiting from YARN containers or running Hadoop over Docker. Their solution was to use Docker containers. Altiscale got acquired by SAP HANA group. SequenceIQ is another startup company that attempts to solve the same problem; it got acquired by Hortonworks. Such computationally demanding activities are usually carried out by multiple engineers simultaneously, especially during peak periods close to project deadlines. All of these considerations make it critical for research and development (R&D) institutions to provision HPC resources, plan job scheduling, and do smart resource allocations.

With the advent of Serverless applications and embedded Fog Computing tools such as OpenFaaS [57] and their public cloud, backend counter-parts such as Amazon Lambda, and Microsoft Functions, and Google Cloud Functions [58] it has recently become feasible to push some of the computations to the edges, i.e. Edge Computing.

Increased sensitivity of end users to product security and privacy will determine their choice of the aforementioned cloud deployment model: public, private or community. The most sensitive users with top-secret products to analyze will deploy private clouds behind firewalls [59]. No information should leak outside unless their security policies are breached from outside or inside. However, these users will potentially trade-off cost savings and compromise high utilizations [60]. Publicly deployed cloud services are potentially prone to both insider (i.e. service provider) and outsider (i.e. man-in-the-middle) attacks. Using the well-known public and private key encryption methods (e.g. SSL used in HTTPS) as well as applying Message Authentication Codes (MAC) over the exchanged FEM models can alleviate some of these problems, namely data security and integrity will be protected. However, these security precautions should not hinder sharing of data among cloud users. One way to allow controlled sharing is to use Authorization-Based Access Control (ABAC) methods [61] instead of Role-Based Access Control (RBAC), where the former allows finer-granularity control over the resources. In ABAC, users can assign certain usage rules or quotas (e.g. using SAML certificates [61]) over server, storage, and network resources or over data stored in file systems and delegate these privileges to other users. A privilege works for only a given resource and provides limited access. RBAC (e.g. username/password and group) allows unlimited access to all resources owned by a user/group. Therefore, even a small inadvertent mistake done at the user level or at a higher privilege level (e.g. root or administrator) can result in the exposition of significant amount of personal or organizational data.

An advantage of working in the FEA cloud services area is that users are allowed to pre-process their models locally and only upload transformed matrices into the cloud service for further processing. This way even a potential malicious attacker inside the service provider would be no able to reconstruct the design details of the original customer product. Basically, after these transformations one cannot differentiate whether the matrix belongs to the chassis of a stealth plane or a toilet pump.

Another related but orthogonal issue to security of cloud services is the reliability and availability of these services. Since these are well-matured research areas no further details are given here, but refer users back to publications on data redundancy and system fault-tolerance [62]. Denial of Service (DoS) attacks are yet another concern for cloud service availability, but again there are well-known network filtering and throttling techniques to eliminate (or alleviate the adverse affects of) these types of attacks.

Due to the government compliance and security privacy issues people in different countries may not disallowed to use international FEA services. Therefore, FEA services in different parts of the world may still be a valuable proposition. When standards such as Open Grid Forum's Open Cloud Computing Interface (OCCI) [63] are developed these providers will be able communicate and share/port models among each other easily.

CHAPTER III

DESIGN AND IMPLEMENTATION OF SOFTWARE LAYER

3.1 FEA Service Architectural Design

A wide variety of sectors including automotive, aviation, construction, and academia deal with mechanical structural analysis problems. In these sectors a rigorous mechanical evaluation of components has to be carried out before they are produced. This practice saves time and money in the design, prototyping and manufacturing phases of a product's lifecycle [64] and increases the reliability of the produced parts reducing the possibility of recalls and critical failures [65]. In addition, different parts of a complex system (e.g. engine, tires, wings, and chassis of a plane) are usually designed by different groups or subcontractors in different parts of the world. Therefore, a FEA cloud service could facilitate both independent and collaborative parts design and development processes.

In today's practice engineers first use CAD tools for quick and accurate parts' design. Next, they save their designs in proprietary file formats (e.g. CATPart, prt, dwg) or export these files in portable formats such as initial graphics exchange specification (IGES) or standard for the exchange of product model data (STP) [64]. Currently the "STL" format designed for rapid 3D STereo-Lithographical prototyping to provide surface geometry information is imported. To obtain a realistic Finite Element Model (FEM) from the CAD file, a pre-processor tool (such as NetGen [66]) can be used to import the design, apply meshing to it, select materials for the part, set boundary conditions, and define external forces. The extended model is then saved in a special file format that can be processed by the FEA solvers.

Fig. 3 shows the architectural design and some of the implementation details of the FEA service. It consists of the web portal, pre-processor, job scheduler, solvers, and post-processor components in their respective order of execution.

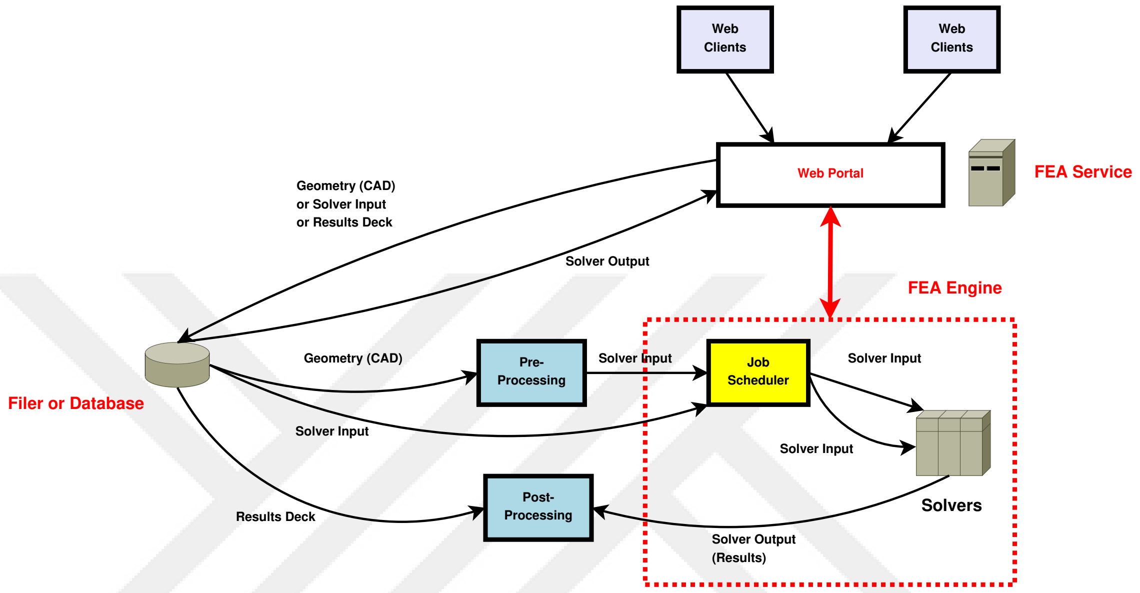


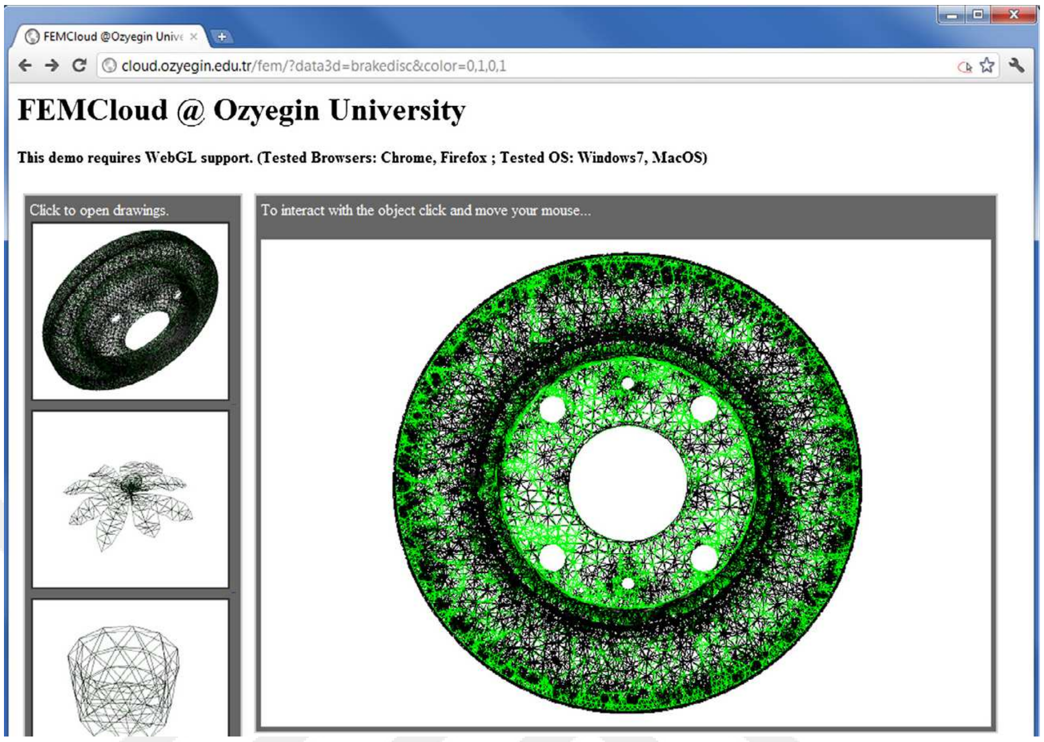
Figure 3: FEA Cloud Service Architecture.

3.1.1 Web portal

Web portals such as Liferay, Drupal, and Joomla [67] serve as the front-end for all user-to-cloud-service and user-to-user interactions. These interactions include creating accounts, logins, uploading and sharing files, pre-processing and post-processing FEM, communicating results to other users, short messaging, attending forums, blogs, wikis, etc. Each user gets its own account and a private file storage area in the filer or database via this web portal. The files uploaded can be raw CAD files or pre-processed text based solver input files. The interaction is similar to cloud services such as an online email system, but FEA portal also allows users to execute analysis of their jobs on top of the FEA engine. Currently, the Java-based Liferay portal is being used because of its ease of integration with other web technologies and the other components of the FEA service. The portal also allows users to interact with different parts of the FEA engine.

3.1.2 Pre-processor and solver

CalculiX [38] is used as the solver for the online FEA service, because of its open-source availability, wide-adoption in the community and extensive support for solving different engineering problems. CalculiX package has a separate pre-processing tool called CGX (CalculiX GraphiX) [68] that can be used to read and transform the contents of various portable CAD files into a finite element model. In the service design, the pre-and-post processing steps are allowed to be done either (1) offline with desktop tools such as NetGen [66], FEMAP and CGX, or (2) offline inside the web browser's JavaScript [69] engine (such as Google Chrome V8) for quick interactions or privacy, or (3) online through the use of custom JavaScript integration code for WebGL backed by a server side meshing engine (e.g. NetGen API running on the servers). Note that in the last two cases no extra software installation will be required on the client side and in case (3) even large-scale meshing jobs can be done quickly with high-end servers. "WebGL is a cross-platform, royalty-free web standard for a low-level 3D graphics API based on OpenGL ES 2.0, exposed through the HTML5 Canvas element as Document Object Model interfaces [70]." Fig. 4 shows a screenshot of the 3D viewing of a meshed structure inside the portal of the web site. Canvas element together with the WebGL API can enable the users to interact with (select, rotate, zoom, etc.) the 3D objects especially in the pre- and post-processing phases of the design.


FEMCloud @ Ozyegin University

This demo requires WebGL support. (Tested Browsers: Chrome, Firefox ; Tested OS: Windows7, MacOS)

FEMCloud @Ozyegin Unive
cloud.ozyegin.edu.tr/fem/?data3d=brakedisc&color=0,1,0,1
Click to open drawings.
To interact with the object click and move your mouse...

Figure 4: A screenshot from the online pre-processing step for the FEA service. Meshed structures can be generated from CAD files and viewed online. Note: WebGL is currently supported by Google Chrome, Mozilla Firefox and a few other web browsers, but its adoption is increasing. Certain OS and browser settings may be required. See http://cloud.ozyegin.edu.tr/fem.

The finite element model is consequently converted into a large sparse matrix by CCX (CalculiX CrunchiX) [38] representing the system of linear equations and solved by the underlying solvers such as SPOOLES (Sparse Object-Oriented Linear Equation Solver) [71]. Results obtained helps to estimate the physical displacements, stresses and strains on the structure under applied forces. Several other open-source or proprietary linear equation solvers (PARDISO [72], TAUCS) can also be used together with CalculiX [73]. SPOOLES direct solver is used in this thesis; therefore the details for other solvers are skipped for brevity. There are also tools for sub-structuring objects before executing the FEA such as METIS and its parallel version

PARMETIS [74]. METIS is used for partitioning graphs and finite element meshes, and producing fill reducing orderings for sparse matrices. A sub-structuring (aka domain decomposition) tool is not included in the design for two reasons: (1) research shows that [73] parallel equation solver methods that work at a lower-level than the FEM can be much faster than parallel sub-structuring methods, (2) Sub-structuring requires explicit knowledge about the geometry of the object: As will be seen in Section 3.4 customers can be sensitive about the privacy of their design and the fact that the cloud service provider knows about their intellectual property can be a big concern.

Solving the equation in the matrix form $[K].\{u\} = \{f\}$, is essential in both linear and nonlinear, static and dynamic FEA [73]. In the context of structural mechanics, $\{u\}$ is related to the displacements of each finite element. SPOOLES has four major calculation steps:

- Communicate: Read $K\&f$ matrices,
- Reorder: $(PKP^T).(Pu) = Pf$,
- Factor: Apply Lower-Upper (LU) factorization,
- Solve: Forward and Backward substitutions.

SPOOLES can be executed in a serial (single-threaded), multithreaded (pthreads) or multi-node (MPI) fashion [71], therefore all of the steps above can be parallelized. The results (displacements and stresses) are saved in a specially-formatted file called FRD in CalculiX.

3.1.3 Post-processor

Post-processing can also be done online or offline similar to preprocessing. For example, the CGX tool can be used to read the FRD file and visualize the results on the object under given forces as shown in Figure 5.

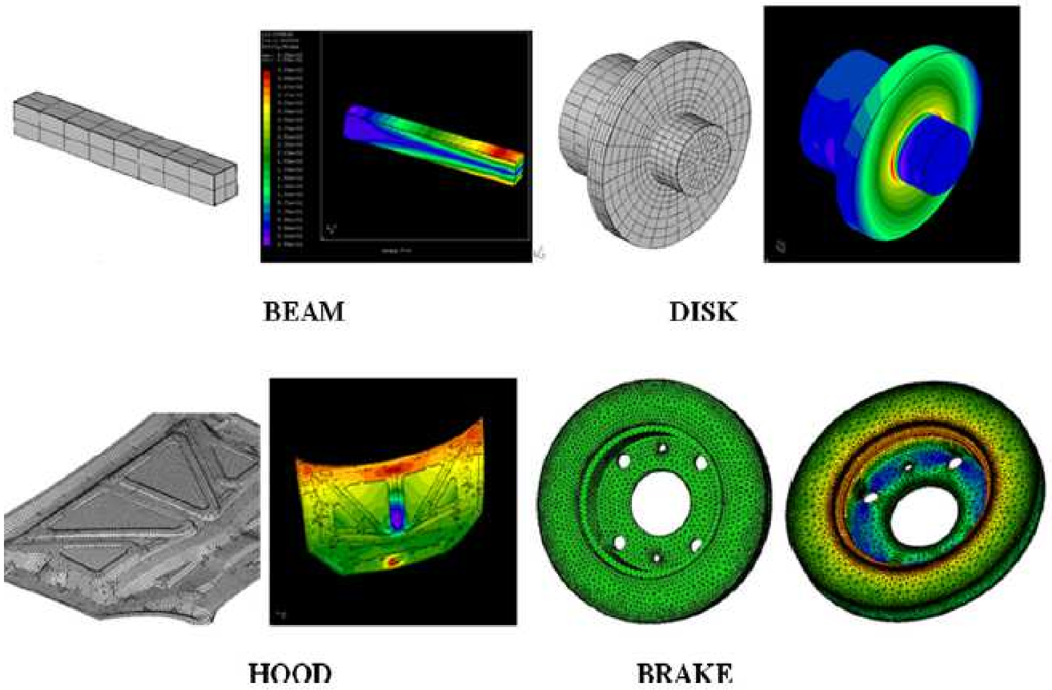


Figure 5: Screenshots from FEA of Beam, Disk, Hood, Brake objects under dynamic forces.

After all results are saved into the file system in FRD file format, CGX tool can be used to visualize the stresses and strains on the object under given boundary conditions as shown in Figure 3.

3.1.4 Job scheduler

FEA jobs with different CPU, memory and I/O needs need to be first characterized and then scheduled accordingly for optimal processing performance. In addition, multi-tenant cloud services such as ours require a careful balance between job isolation for customer quality of service (QoS) assurance and mixed execution for high throughput and better resource utilization for service providers. This is a multivariate optimization problem that can be mapped into an NP-hard "bin packing" problem. The scheduler needs to make automated, smart decisions on admission control, job throttling, concurrent scheduling and even rescheduling. The evaluations and results of different representative FEA loads on single-core, multi-core and many-node (MPI) configurations on two alternative systems (low-end PCs and high-end servers) are presented and different scheduling techniques are discussed in the following sections.

The next section describes some of these representative workloads and discusses some of the job-to-resource scheduling issues based on real results.

3.2 Workload Characterization

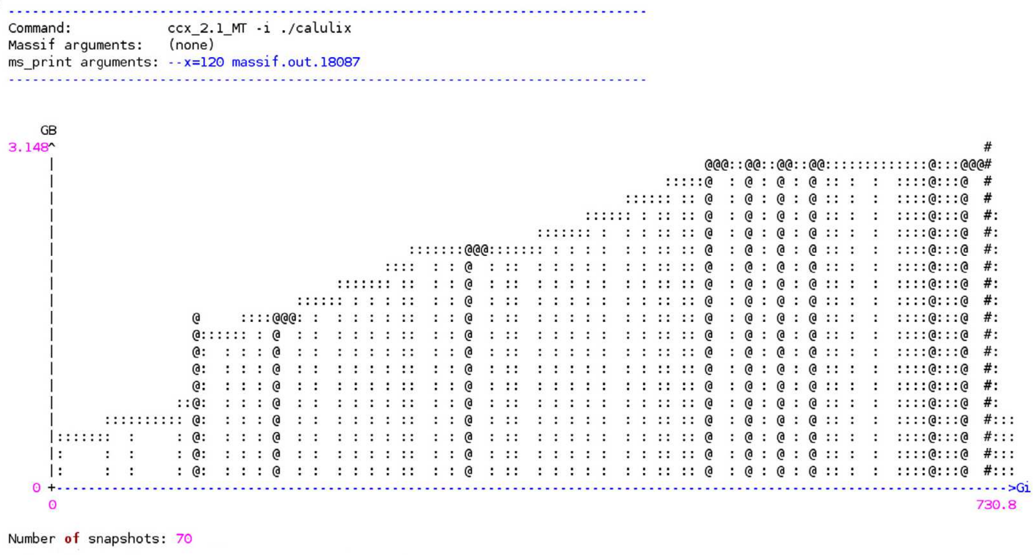
In this section, the three models shown in Figure 5 are used to guide the performance tests and the FEA service design. These models are chosen because of the differences and some controlled similarities in their processing complexities. The first is an $8m \times 1m \times 1m$ concrete cantilever beam under a 9 MN bending force applied at its free end (i.e. a civil engineering case). The second is a jet engine disk under a high-speed centrifugal force (i.e. an aviation case). The third and fourth are cases from the automotive industry; first being a car Hood that is getting loaded with a concentrated force from above and second being a Brake rotor under centrifugal forces. Both the pre-and-post processed versions of these structures are shown in Fig. 5. Red tones represent the maximum stress areas in the body and show potential points of failure [68]. The product designers are expected to evaluate these results and either alleviate the stress points via redesign or indicate conditions for acceptable use of their products in their data sheets.

The initial file size of these models is relatively small (largest hood is $<$ 6MB) and therefore they can be immediately mapped to memory resolving any further disk I/O issues. When meshed at a very fine-granularity the file sizes can go up to a few GB increasing the overall impact of I/O and requiring a more careful consideration. Other processing-related FEM parameters include number of elements (hexahedrons, tetrahedrons, etc.), integration points for each element, and the number of nodes (or total points) in the model. Most of these parameters are found that they do not have a major effect on the performance, since the model is transformed into matrices before being processed by the solvers and most of these matrices are extremely sparse (i.e. filled with zeros). Sparse direct linear equations solvers such as SPOOLES can take advantage of this fact to obtain a compact, memory-efficient representation of the FEM. Therefore, in the results (Section 3.3), it is observed that both the computational complexity and the memory requirements for the mechanical structural analyses done in this thesis are positively correlated with the number of non-zero elements (NZE) in the matrix (e.g. see Fig. 7), which was also indicated by prior related work [75]. However, note that there can be counterexamples to this rule for FEA of objects with drastically different geometries, materials and analysis types. Section 3.3.7 discusses one such scenario for the effect of geometry on performance.

The future work includes handling parallel I/O for bigger FEM files with MPI-IO or using distributed task processing systems such as MapReduce [76] for this purpose. MapReduce is used in other cloud projects to process 100s of GB of enterprise log files and therefore it is possible to apply it to parallelize FEA I/O loads as well.

The NZE count and sparsity (i.e. % NZE/Total Elements) of the sample objects can be summarized as follows: Beam $38K$ NZE (7.4%), Disk $2.96M$ NZE (0.2%) and Hood $26.2M$ NZE (0.013%). The NZE of the Brake component is varied from 8 to 55 million via controlled fine-granularity meshing and its effects on memory and CPU time are measured in Section 3.3 (see Table 7).

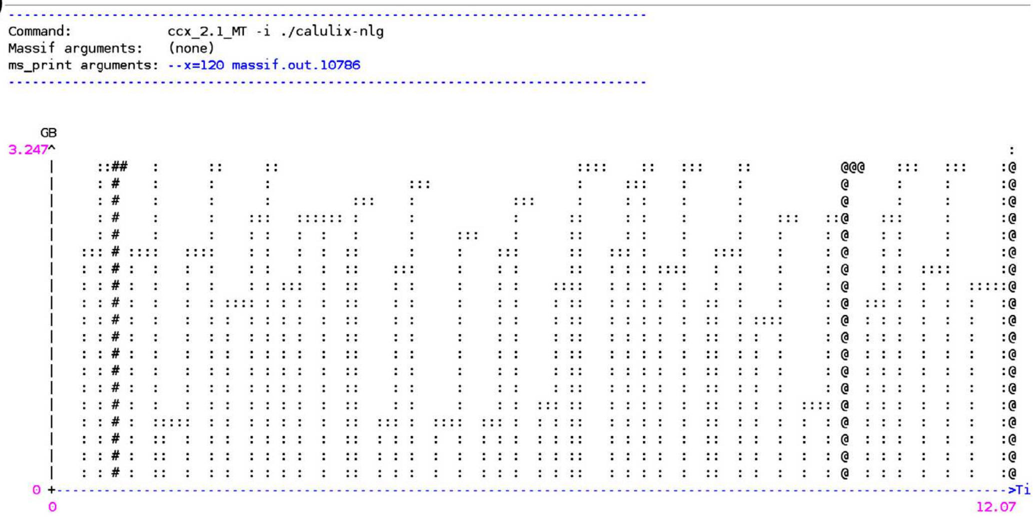
(a)



```
--------------------------------------------------------------------------------
Command:            ccx_2.1_MT -i ./calulix
Massif arguments:   (none)
ms_print arguments: --x=120 massif.out.18087
--------------------------------------------------------------------------------
```

```
Number of snapshots: 70
```

(b)



```
--------------------------------------------------------------------------------
Command:            ccx_2.1_MT -i ./calulix-nlg
Massif arguments:   (none)
ms_print arguments: --x=120 massif.out.10786
--------------------------------------------------------------------------------
```

Figure 6: Memory profiling and comparison of linear and nonlinear FEA of Hood using Valgrind Massif tool. Nonlinear analysis of Hood did 18 iterations. Gi–Ti refer to Giga and Tera instructions, respectively. Results for FEA of other objects looked similar and therefore they are omitted for brevity.)

The parallel portion of the analysis code also affects its processing performance over multiple cores and nodes. Amdahl's law dictates that adding more cores beyond $8-16$ to solve $50-75\%$ parallelizable jobs, which are very common in FEA, will have a small incremental performance impact [77]. These FEA jobs are processed on two different systems: high-end servers each with 8-core CPUs and 12–24 GB memory and low-end PCs with 2-core CPU and 2 GB memory. The code is timed and the parallel portion is measured to span 65% and 61% of the overall execution time, respectively. The results confirmed the validity of Amdahl's Law (i.e. increased benefits with diminishing returns) for these CPU-intensive HPC loads and therefore the details are skipped for brevity. However, cases where some large NZE jobs triggered swapping (with kswapd in Linux) due to lack of memory are also encountered, especially on the PC system with less memory. In such cases, the jobs will eventually complete, but it will be impossible to predict when they will or what the overall system throughput may be. Therefore, such cases should be avoided. Section 3.3 will present detailed experimental results.

3.2.1 Linear vs. nonlinear analysis

The linear analysis theory is based on the assumption that the displacements are small with respect to the geometry of the structure and the material is linear elastic (Hookean). Therefore, the solution is found in one step. This assumption is no longer true when the displacements are large so that the applied forces also affect the geometry and the equilibrium equations must be iteratively reimposed during the computation, or the material behavior is nonlinear [78].

The nonlinear analysis divides the problem into smaller incremental steps and the final solution is found by iteration and by checking convergence conditions. The size of increments can be defined by the user, but in CalculiX it is advisable to let the program decide on this parameter at run-time for faster convergence. The convergence criterion is that the residual forces (i.e. difference between the internal and the external forces) of the structure are small enough (e.g. ¡0.00001%). If they are small enough, the solution is accepted as converged. Otherwise, the iteration will continue until either convergence or a maximum iteration count is reached. If the maximum iteration is reached, the computation will stop.

3.3 Experiments

In this section, the experimental setup for cloud performance evaluation is described (Subsection 3.3.1) and a systematic analysis of the described FEM loads is presented. Since the proposed FEA service will be a multi-tenant, concurrent job processing system, there is a need to understand the CPU usage and memory impact of executing different types of jobs (different FEMs and linear vs. nonlinear analysis) in a multi-threaded fashion. Second, the limits of concurrent processing are tested by scheduling J identical jobs each using a single thread (Subsection 3.3.2). Third, the number of threads T for each job is increased to better utilize the C cores in each node (Subsection 3.3.3). Then, jobs that cannot be sustained on a single node are run over multiple nodes using MPI to benefit from additional distributed computing resources (CPU and memory) (Subsection 3.3.4). Next, different jobs are mixed in order to compare different scheduling techniques and advantages of smart scheduling that takes adaptive parameter tuning into account are shown (Subsection 3.3.5). Finally, the effects of structure geometry on the analysis performance are investigated (Subsection 3.3.7).

3.3.1 Experimental Setup

Two types of systems are used for the experiments. The first is a PC cluster consisting of $8 \times HPDC5850$ personal computers (PCs) with one 2.3 GHz AMD Athlon X2 Dual-Core Processor, 2 GB Memory and 250 GB 7200 rpm disk. These nodes are connected via a 100Mbps Cisco Catalyst 2950 Ethernet switch. The second system is a high-performance IBM Blade system in the data center with $4 \times IBMH22BladeServers$ in HPC-H chassis providing 1Gbps connectivity among the blades. Each IBM blade server has two 2.40 GHz Intel Xeon Quad-Core E5530 Processors, 12–24 GB Memory, and two 72 GB 15000 rpm disks each. To summarize, PCs have 2 cores and servers have 8 cores per node. Red Hat Enterprise Linux 5 server OS, CalculiX, SPOOLES, and openMPI are installed on all of these systems. Four FEA workloads described in the previous section are used for performance comparisons, namely Beam, Disk, Hood and Brake and their mixes.

3.3.2 Single S/W Thread and OS Scheduler

In this subsection, a controlled experiment is executed by running J identical jobs of the three workloads (Beam, Disk, Hood) concurrently on a single server node and a single PC node. The software multi-threading (MT) parameter is set as $T = 1$. Figure 7 shows the execution time and throughput results in a $\log_{10}$ - $\log_2$ scale of time and job count parameters, respectively for the two different hardware settings. Results confirm that the server (with 8 cores) and the PC (with 2 cores) can respectively handle 8 and 2 concurrent Beam and Disk jobs without any performance decay. No latency increase per increasing job count is observed until the physical core count is reached, C. In the sustained performance (i.e. flat latency) regions it is found that the time difference between the loads to be directly proportional to the number of NZE (38 K vs. 2.96 M vs. 26.2 M). Based on these observations, it is possible to better predict the expected run time and memory needs of newly arriving jobs and mixes of jobs by comparing their NZE with those of the already characterized jobs and their observed performances. However, one has to be careful about making exact decisions based on NZE as there are other factors affecting the performance.

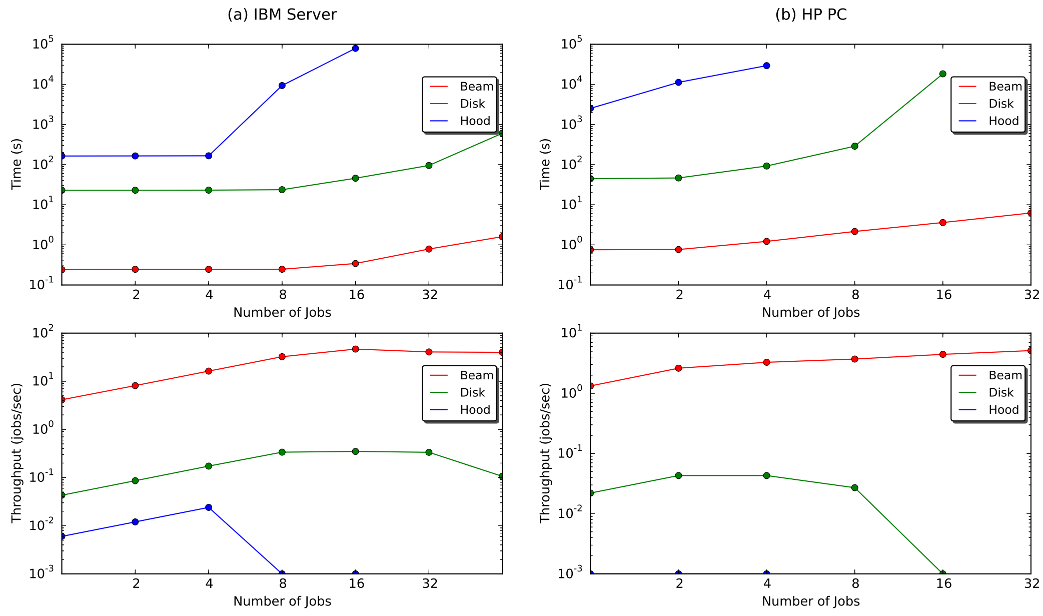


Figure 7: Processing of beam, disk and hood jobs with only 1 thread/job on (a) IBM server and (b) HP PC systems. The time increases and throughput drops immediately when the total core count is exceeded.

The Hood job sees significant performance degradation after four jobs in server (with 12 GB RAM) and even with one job in PC (with 2 GB RAM). The reason is simply the lack of enough RAM to sustain the processing of multiple concurrent Hood jobs. Memory profiling using Valgrind Massif profiler [79] has been carried out and it is found that the maximum memory requirements of Beam, Disk and Hood to be 4.7 MB, 380 MB and 3.15 GB, respectively. These values were all approximately 0.13 KB times the NZE of these model matrices and since the object structures were quite different the result is surprising. This magic value could be explained as the total size of global variables (structs, integers, etc.) that need to be allocated in CalculiX and SPOOLES per NZE. The subject with different multi-threading parameters, linear-vs.-nonlinear analysis and with objects that have larger NZE (e.g. Brake) are investigated in Subsection 3.3.5.

Figure 6 shows the dynamically-changing memory allocation of CalculiX for the Hood job over time for linear and nonlinear analysis with 1 thread. The linear analysis makes one cycle first allocating the memory for the matrix, factorizing and solving the problem and finally de-allocating it when the calculations are finished. The maximum memory allocation is 3.148 GB. The nonlinear analysis will go into a cycle of memory allocation and de-allocation for each iteration, but the maximum allocated memory will be comparable to the linear analysis. For example, the nonlinear analysis of the Hood job iterates 18 times before convergence and the peak memory value is around 3.247 GB (within 3% of linear analysis).

Next, the same experiments are repeated with different thread counts (1–2–4–8 threads) on each model. It is found that increasing the number of threads slightly increases the maximum memory used (see Table 1). The percentage of memory increase depends on the job. The increase is mainly due to the replication of application code (e.g. ccx program) in memory and not the matrix as this data is shared among the threads. It is also know that Linux OS will not replicate the shared libraries (such as libc.so and ld.so) for threads. An investigation of memory maps (cat /proc/processid/ maps file in Linux) reveals this fact. Note that if jobs are distributed with MPI on multiple machines the benefit gained from data and library sharing will be forfeited.

Table 1: Increasing the multi-threading count slightly increases the maximum memory used while processing FEA jobs (Hood on IBM server).

Threads	Max Memory (GB)	KB/NZE
1	3.148	0.126
2	3.154	0.126
4	3.198	0.128
8	3.315	0.132

Referring back to Fig. 7, the throughput results show that for short-running jobs (i.e. Beam) there can be a small incremental benefit in pushing more jobs than the core count, C, of the system, but with the long-running jobs the throughput can only be sustained until $J \leq C$. A significant performance difference between the time and throughput results of the server (3– 10×) and the PC is observed. This can be attributed to the differences in CPUs, bus speeds, and memory sizes. It can be concluded that the cheap PC clusters may not be feasible for provisioning a high-quality FEA cloud service. The big performance difference between the two alternative strategies (e.g. the 3–10× more jobs completed at the same time) justifies the higher price of servers.

3.3.3 Multi S/W Thread and OS Scheduler

In this subsection, the multi-threading (MT) parameter, T, is increased at the application level to see whether any benefits can be gained by adapting this parameter. MT creates a potential for running parts of a job on different cores. Note that the OS will not be able to parallelize jobs that are not explicitly multi-threaded by the application. However, one cannot easily specify on which core each thread should execute explicitly in a single machine, since this is dynamically decided by the OS scheduler (this matters less when cores are symmetric). The motivation for automatically tuning T parameter is two folds: (1) long running jobs will be parallelized (2) mixes of jobs can be better interleaved. All results in this section were obtained on the IBM systems.

Fig. 8 shows the total execution time and throughput results for the Beam job for different T (1–8) values. As Beam is a very short running job, trying to divide it further into pieces has a diverse effect on the performance (execution time increases and throughput drops). The best throughput is obtained at $T = 1$ and $J = 16$ on IBM server ($C = 8$). Pushing more jobs beyond this point also degrades the performance.

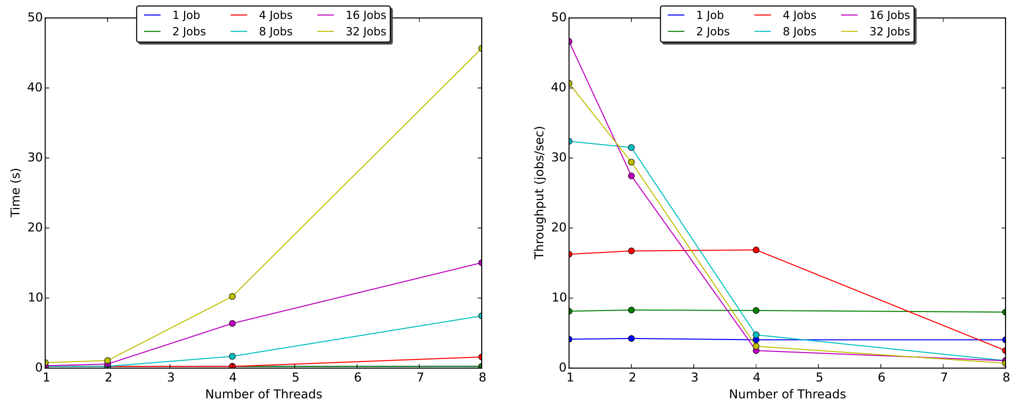


Figure 8: Execution time and throughput results for many-task multi-core processing of beam on IBM server.

Figure 9 shows the time and throughput results for Disk. Note that the latency drops and throughput increases until $T = 8$ for $J = 1$. For $J = 2$ throughput increases until $T = 4$ and then drops slightly. For $J = 4$, throughput increases until $T = 2$. By observing this trend, the rule $J \leq C$ from Subsection 3.3.2 can be extended as $J \times T \leq C$. The throughput of Disk workload is highest at $J = 16$ and $T = 1$ point (0.35 jobs/s). However, note that the total latency is also high around 50s as $16 \times 1 > C = 8$. Therefore, it is again found the best latency-throughput trade-off at $J \times T = C$ point with $[J = 8, T = 1]$ and $[J = 4, T = 2]$ points. The former choice gives better throughput and the latter has better latency. The choice can be made based on how critical either performance metric is in a given context. For example, the service providers would prefer higher throughput, but the clients would prefer lower latency.

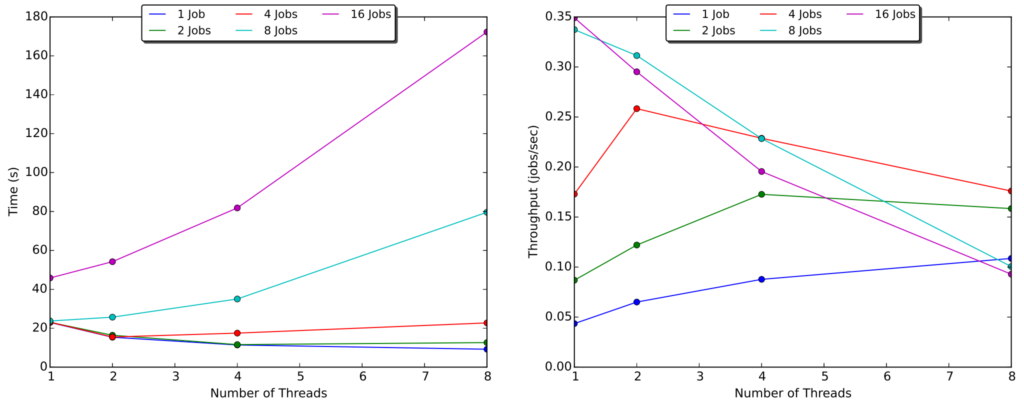


Figure 9: Execution time and throughput results for many-task multi-core processing of disk on IBM server.

The results in Figure 10 further confirm the findings using the Hood job. The highest throughput/latency point is at $[J = 2, T = 4]$. To conclude, longer-running jobs will benefit significantly from increased (but carefully-controlled) software threading, until the core count value, C, is reached in a single node. However, for mixed jobs the tuning and multi-node distribution will have to be done automatically. In current state-of-the-art, programmers usually assign jobs the maximum threading parameter that (they believe) will fully-utilize the allocated resources. This is called as the “aggressive strategy” and compared to the “smart scheduling” in Subsection 3.3.5.

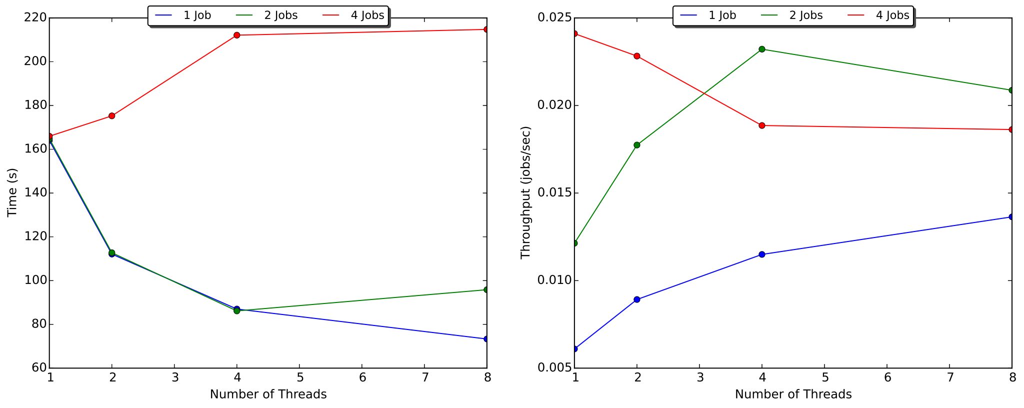


Figure 10: Execution time and throughput results for many-task multi-core processing of hood on IBM server.

3.3.3.1 Multi-threaded nonlinear analysis

Since nonlinear analysis is an iterative process, the effect of multi-threading is tested on nonlinear FEA and compared it to linear FEA with one of the jobs. The results in Table 2 were obtained on the IBM server with 24 GB RAM and using the Hood job. Results show that nonlinear jobs can gain more speedup from multi-threading as the percentage (and importance) of the initial serial part of the code diminishes as there occurs more iterations over the same code. Note that each iteration in nonlinear analysis (which maps approximately to one linear FEA) will internally have a serial and a parallel portion. That serial portion cannot benefit from parallelization and therefore the speedup is not as high as expected. The increased speedup values in nonlinear analysis can be attributed to the reduction of the importance of the global initialization phase. In this run the analysis iterates 18 times before convergence and iteration count depends on various factors including convergence criterion, object geometry, and applied forces. If the maximum iteration count is set to a relatively small value such that it is reached before the highly-constraint convergence criteria (e.g. $< 0.000001\%$), then the total non-linear analysis time can be approximately calculated as $linear\_time \times iteration\_count \times speedup\_ratio$ between linear and non-linear analysis. Also note that this is a property of the direct solver and iterative solvers may demonstrate different behavior. Finally, the nonlinear analysis in Table 2 also consistently shows performance drop after thread count (16) exceeds the core count (8).

Table 2: Comparison of the time and speedups for linear and nonlinear analysis of Hood on IBM server.

	Linear		Nonlinear	
Thread	Time (s)	Speedup	Time (s)	Speedup
1	160.24	1	2517.218	1
2	113.2	1.41	1833.705	1.37
4	94.82	1.69	1220.27	2.06
8	83.38	1.92	983.537	2.56
16	97.77	1.64	1057.107	2.38

The nonlinear time analysis for the Hood and Disk jobs are completed with different concurrent job and thread counts. The results further confirm the $J \times T \leq C$ rule.

Note that some of these problems are open-ended with respect to the overall FEA cloud service design as other factors including accounting and quotas will also determine how long a certain user can be allowed to run a job on the cloud system. In a certain accounting model, jobs may be preempted and paused (or killed) because the user has no remaining credit for analysis.

3.3.4 MT vs. MPI, Single and Multiple Nodes

Table 3 shows the comparison results for software multi-threading (MT) and the Message Passing Interface (MPI) in a single machine for one hood and one disk job (J=1) on the IBM server. It is known that MPI (distributed memory model) will communicate via explicit message passing and in MT threads will share the same process address space (shared memory model). First, the overhead associated with messaging is quantified irrespective of any network equipment capabilities (i.e. in a single machine). Table 3 shows the comparison results for software multi-threading (MT) and the Message Passing Interface (MPI) in a single machine for one Hood and one Disk job (J = 1) on the IBM server (with 12 GB RAM). Results confirm that as the core count increases, MPI performance degrades over the MT version for both jobs and the slowdown reaches $\sim$ 0.82 level (i.e. 18% performance loss) even at 8 cores.

Table 3: Time comparison of MT vs. MPI on IBM with Hood and Disk jobs.

	Hood			Disk		
Core	MT	MPI	Slowdown	MT	MPI	Slowdown
1	158.9	161.2	0.99	22.71	22.87	0.99
2	121.2	126.8	0.96	15.36	18.33	0.84
4	85.44	100.6	0.85	11.57	14.01	0.83
8	73.3	83.83	0.87	9.286	11.3	0.82

Related publications suggest designing new processor architectures called Message Passing Processors [80] which are distributed memory multi-core processors that communicate using message passing among themselves. However, the outcome suggests that for tightly-coupled systems shared memory architectures or “MT” should be preferred over “MPI”. MPI should only be preferred as an option to “scale out” resources beyond a single node as there are always limits to “scaling up” local resources. For example, it is seen that the Hood job could not run properly on a single HP PCs ($N = 1$) due to lack of memory. Table 4 shows the benefits of employing MPI in such scenarios. If MT and MPI are taken as two alternative strategies for resource pooling (or virtualization), the same total number of cores, e.g. $TC = 2$, can be configured either as one node [$N = 1$,$C = 2$] (using MT) or two nodes [$N = 2$,$C = 1$] (using MPI). While Table 3 showed some performance degradations due to messaging, Table 4 shows that there may still be large savings with MPI due to the avoidance of memory swaps (e.g. first case in Table 4, 1325.6/475.5 $\sim$ 2.79$\times$ gain). Similar gains are observed for the TC = 4 [2 $\sim$ 2 and 4 $\sim$ 1] and TC = 8 [4 $\sim$ 2 and 8 $\sim$ 1] configurations over MT, but with diminishing returns as the job does not benefit from the additional memory resources. Note that the network bandwidth for this PC cluster was 100 Mbps and this may be the cause of some of the bottlenecks. In the future, there is a plan to repeat these experiments with 1–10 Gbps switches.

Table 4: Execution time comparison of MT vs. MPI for Hood on PC.

Total Core	Node x Core	Time (s)
2	1 x 2	1325.4 (Swap)
2	2 x 1	475.5
4	2 x 2	278.5
4	4 x 1	268.2
8	4 x 2	318.0
8	8 x 1	271.0

3.3.5 Smart Scheduling for Mixed Jobs

In previous subsections, each workload type (i.e. the Beam, Disk and Hood) is evaluated extensively, but separately, on different number of cores and with varying threading (T) and concurrent job count (J) parameters. In this subsection, these three workloads are mixed to get one step closer to a real life batch processing scenario. Given the processing limits of the systems in Fig. 7 and findings in previous subsections, a job mix that would substantially load the system is created and the effects of different scheduling strategies are highlighted. The mix consists of 4 Hood, 32 Disk, and 1024 Beam jobs and the goal is to finish the batch as quickly as possible. Three different scheduling strategies that are compared are called the Shortest-Job-First (SJF), Aggressive strategy and smart scheduling. SJF strategy simply allocates 1 Thread per job ($T = 1$) and lets the OS scheduler handle the effective job-to-core placements. The Aggressive strategy wants to utilize all cores and splits all jobs into threaded parts as many as the core count ($T = C$) of nodes before handing over to the OS scheduler. The smart scheduler calculates and sets a different threading parameter for different job types based on a logarithmic function of the NZE, predicts the potential memory use and tracks resource usage as well as available capacities. All schedulers use Round-Robin (RR) algorithm for load balancing over multiple machines at this time.

Algorithm 1 Smart Scheduler Algorithm

```
 1: Determine optimum core counts
 2: Determine necessary memory needs
 3: Set priority value to nonzero elements
 4: Place the job into priority queue
 5: if There is sufficient memory in any of the nodes and enough cores in them then
 6:    if There is only one maximum memory granting node then
 7:       Select maximum memory granting node
 8:    else
 9:       Use round-robin technique
10:    end if
11: else if There is single no single node with sufficient memory or core then
12:    Use MPI to distribute the job with possible minimum node count
13: else
14:    Wait for the next job completion and check once again
15: end if
```

The smart scheduler briefly works as follows:

1. The jobs are characterized when they are uploaded to the system. Optimum core counts and necessary memory needs are determined (Characterization).

2. When a job is submitted for processing it is placed into a priority queue with its priority value set to its amount of nonzero elements (use benefits of SJF).

3. If there's enough memory in any of the nodes and enough cores in them, then the maximum memory granting node is selected. When there are multiple matching nodes, round-robin technique is used (implicit load balancing).

4. If there is no single node with sufficient memory or core available for the job, MPI is used to distribute the job with possible minimum MPI node count (avoids swapping).

5. If there's no resource available, then the new job waits for the next job completion and checks once again (admission control). If the job requires more memory and CPU cores than the cluster can provide than policy-based admission control will be applied (either all rejected, or some admitted based on customer class type).

The new jobs arriving into the system would be characterized and classified accordingly by the Smart scheduler before execution. For example, its sets the threading parameters as T=1 for beam, T=2 for disk and T=4 for hood. As seen in Table 5, the Aggressive multi-threading strategy creates additional thread handling burden on the OS scheduler slowing down the total execution times while attempting to utilize all the cores. On one server aggressive is 7.53 times slower than Smart and 7.07 times slower than SJF schedulers. Smart scheduler provides 7% additional improvement over the SJF strategy that still depends on a very efficient OS scheduler. For two servers, Smart provides a speedup of 6.79x over Aggressive and 1.53x over SJF. Estimates for execution times for this job mix are also obtained by superposing the concurrent execution times of each job type using Figure 7 and found these estimates to be highly accurate (within 1-10% range).

Table 5: Comparison of Scheduling Strategies over the Mixed Many-Task Job on IBM server.

	Smart (T=Adaptive)	SJF (T=1)	Aggressive (T=8)
1 Server	240.7 (7.53)	256.4 (7.07)	1812.5 (1)
2 Servers	130.45 (6.79)	199.98 (4.43)	886.2 (1)

3.3.6 Design of Smart Scheduler

The main structure of the proposed scheduler can be seen in Figure 11. The flow of a submitted job passes through six phases. First of all the model is submitted to the system. This is the initial submission of the job, it is a raw user input which enters the system. Then the job enters the characterization phase which is a FIFO queue where each submitted job gets analyzed and created. At this point jobs are in ”New” state. As the third phase, pre-processing is applied to the jobs. They get categorized and created. This portion of the scheduler is the actual producer of jobs for the solver. Once the jobs are pre-processed and created their states become ”Ready” and they enter an aging priority queue where they wait for execution. Solver is actually the consumer of jobs. Its main function is select the most appropriate job from the queue, submit it to the cluster and turn its state into ”Running”. After this phase, jobs enter the Terminated Job List where all ”Terminated” jobs wait for post-processing.

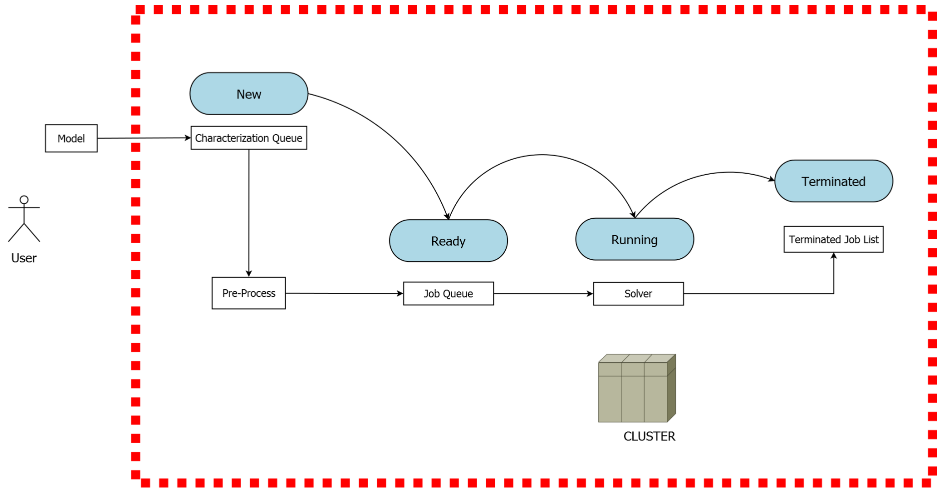


Figure 11: Main structure of the scheduler.

When a job is submitted to the system it occupies the ”New” state and waits for the pre-processor to pick the job for evaluation. After the job is pre-processed, classified and submitted to the queue it occupied the ”Ready” state. The smart scheduler dequeues the job and submits the it to the cluster for solution. This state of the job is called ”Running”. A job may be in the ”Terminated” when its execution is finished. Execution can get terminated in a normal way or can be killed by the user and also can have a solution-time error. These states of a job are represented in Figure 12.

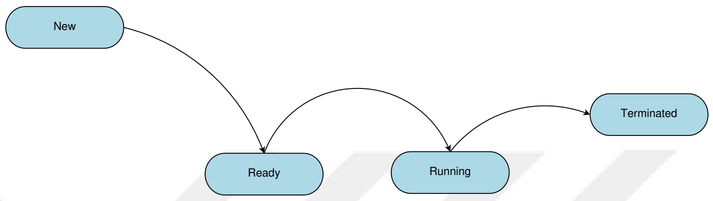


Figure 12: Job State Diagram.

In the heart of the smart scheduler, the classical Producer-Consumer approach is being used. Here the producer module characterizes new jobs and pushes them into the queue. In the smart scheduler module is called the ”Pre-Processor”. The module called ”Queue” actually holds the jobs in ”Ready” state in a heap data structure. Finally there is the Consumer module which houses the main implementation of the smart scheduler described as Algorithm 1. This module aims to pick up the most appropriate job in the queue and submit it to the cluster. This approach is demonstrated in Figure 13.

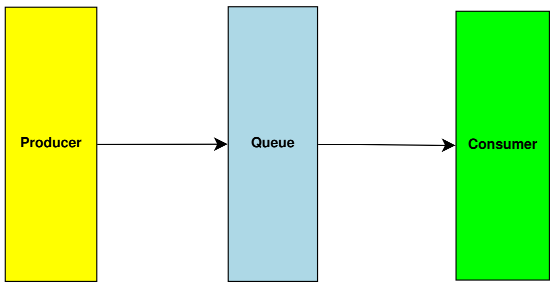


Figure 13: Producer-Consumer Approach.

Pre-Processor (Producer) module tries to determine the non-zero elements of the resulting matrix for a submitted job. During the experiments leading to the design of the smart scheduler, it is observed that the amount of non-zero elements in the sparse stiffness matrices are the main driver behind the CPU-Time, CPU-Count and RAM needs. Therefore, the proposed Pre-Processor module aims to figure out the amount of non-zero elements of the jobs in “New” state and using this information it defines the CPU, Node and RAM needs of a job and makes them “Ready” for the consumer. This parameter (non-zero elements) also gives an idea about the most appropriate solver-preconditioner choice.

As the data structure that holds the jobs that are in ”Ready” state, a Priority Queue which gives jobs a priority between [0:127] is implemented. In this approach, an assumption is made so that the lower the number more prior the job is. In order to avoid starvation issues which rise frequently when priority queues are used, an aging mechanism is introduced. Each job’s prioritization is increased by one after waiting for 15 mins. And finally, if two jobs have the same priority number then the one who has entered the queue before has the priority relative to the newer one.

Solver (behaves in the design as the Consumer) pops up the most appropriate job from the queue and submits it to the cluster. Within this context, “most appropriateness” is a dynamic concept that changes its meaning depending on the status of the cluster and waiting time of the jobs in the queue. Smart scheduler tries to maximize the throughput and minimize the waiting time at the same time using the limited resources of the cluster.

Producer-Consumer implementation uses three threads that are running continuously. The Queue is a blocking-queue that gets locked whenever a thread starts using it. The behavior of these threads are listed below:

- **Producer Thread:** This thread continuously tracks the “New” jobs, pre-processes them and turns them into “Ready”, places into the queue and notifies the waiting consumers (if any).

- **Consumer Thread:** This threads pops up the most appropriate job smartly and submits it into the cluster. If the queue is empty the consumer thread waits to get notified by the producer.

- **Aging Tracker Thread:** This thread continuously checks the ages of waiting jobs and gets them aged when necessary.

These results for smart scheduling are promising. The future work includes completing a comprehensive analysis with different job mixes and comparisons with a wider variety of scheduling algorithms.

3.3.7 Effect of the geometry on performance (with same NZE)

This section shows some of the open research questions in performance characterization of FEA loads. Hsieh et al. [73] found that the geometrical properties of the structure, especially appendages and/or holes, can have significant effect on both the processing time and memory footprint of the analysis. This brought up a very interesting point that triggered further investigation. For comparison, two new models called circle and hollow-circle are generated in this subsection as shown in Figure 14, where the latter is exactly the same as the former object except that it has a hole in the middle. Since the structure with a hole would have fewer elements (assuming the same element type) and thus fewer NZE, the granularity of the meshing for the hollow-circle object is increased until their NZE were approximately the same. Table 6 lists some of the properties and analyses results for these two objects. While the NZE counts are very similar for the two objects (within 2%) the processing times are reduced drastically for the hollow object ($112s \rightarrow 52s$, 54% savings) for 8 threads and ($396s \rightarrow 110s$, 72% savings) for 1 thread. Memory footprint is reduced by 32% for 8 threads and by 45% for 1 thread. The finite element type (C3D4 – tetrahedral with four nodes) used for the two models are also the same. This analysis shows that even the same amount of force applied to two similar objects can have different performance outcomes. All other parameters being about the same, one recognizes a positive correlation between memory usage and the execution time, i.e. the job that uses more memory will also take longer to execute. Note that memory allocation functions such as malloc() also have associated time costs and large FEA loads with huge memory footprints will test the limits of OS memory management. While the Linux slab allocator uses free lists of already allocated data structures (i.e. caches per object type) finding preallocated objects may not be possible when the memory limits are forced. Matrices from finite element problems are often banded, since they describe the coupling between elements and elements are usually not coupled with other elements over arbitrarily large distances in real-life scenarios. Using bandedness of different matrices as a means for performance characterization and comparison among different workloads is left as a future work.

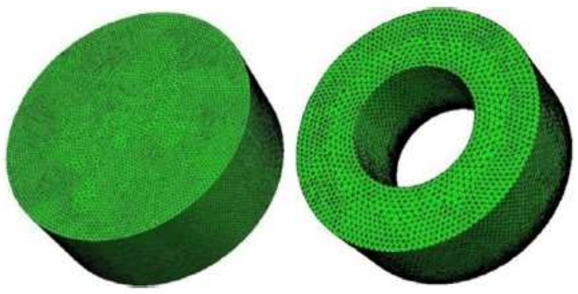


Figure 14: Two control objects, circle and hollow-circle, are used to measure the effect of geometry and force on FEA processing time and memory.

The geometric analysis is continued with the Brake component, shown in Fig. 4, which is also circular and has holes on it. By increasing the meshing granularity of the Brake component, several models called B1–B6 are obtained and listed in Table 7. The NZE count was increased from 8.7 million to 47.1 million. Time and memory usage of FEA for B1–B6 are analyzed. The last model caused memory swaps even on the 24 GB servers, therefore the granularity of meshing is not increased any

Table 6: Effect of geometry on the FEA CPU time and memory usage.

Object	Circle	Hollow Circle
Mesh Options	Very fine, 1, 0.1	Very fine, 1.5, 0.1
NZE	5576731	5469133
Exec. Time (8 threads)	112.33 s	52.43 s
Exec. Time (1 thread)	396.50 s	110.93 s
Memory (%GB 24 GB, T=8)	12.2	8.3
Memory (%GB 24 GB, T=1)	10.8	6.0
Nodes	86344	86937
Element	461,234 (C3D4)	439,169 (C3D4)

further. The memory usage varies from 0.3 to 0.5 KB/NZE. In terms of NZE model B4 is comparable to the previously analyzed Hood job. However, B4 allocates about (10.78 GB/3.15 GB) 3.42× more memory than Hood. The execution time B4 also takes (396.5s/83.4s) 4.75× more time than Hood. Together with the analysis for circle objects above it can be concluded that memory allocation efficiency (time for malloc() and the underlying memory management strategies) can be a significant performance bottleneck in the processing time. This problem is also called the "memory wall" in the OS and Computer Architecture literature. Cloud service designers should also be aware of the architectural compatibilities of all the tools and attached shared libraries (32-bit vs. 64-bit) that their systems are dependent on.

3.4 Discussions on other cloud service related issues

This section provides a detailed discussion on the remaining issues related to the success of a FEA cloud computing service.

3.4.1 Multiple dimensions of smartness

A smart scheduler for the FEA cloud service can utilize various dimensions related to the analysis in addition to the performance related ones mentioned above. Additional dimensions that can be used to make optimal and dynamic scheduling decisions

Table 7: CPU time and memory usage variations for different meshing (thus NZE) configurations of the Brake component.

Model Code	Max Mesh Size (mm)	Non-zero elem. (million)	Memory(%) (for 24GB)	Max. Memory (GB)	Per NZE mem (KB/NZE)	Exec. Time (s) (8 Threads)
B1	2	8.7	9.8	2.549	0.307	83.312
B2	1.5	15.1	17.9	4.554	0.316	-
B3	1.35	19.3	27.4	6.895	0.374	270.254
B4	1.3	25.2	46.6	10.78	0.447	353.162
B5	1.2	37.3	77.2	19.43	0.52	757.04
B6	1.0	47.1	>100	-(>24GB)	SWAP	SWAP

include: class-awareness (jobs from paying vs. free customers), deadline-awareness (using the critical-ratio parameter to decide next job to schedule from a batch), FEA-job-awareness (linear vs. nonlinear analysis, material nonlinearity), file-size awareness (I/O intensive), power-awareness (co-locating jobs on a few nodes to shut-down or spin-down other servers), Amdahl-awareness (estimating parallelizable portion of the code to adjust T parameter), memory-awareness (prioritizing memory needs over CPU in RAM-scarce systems), network-awareness (MT vs. MPI tradeoffs), and last but not least accuracy-awareness (trading-off speed vs. accuracy, e.g. to quickly generate approximate results). In this thesis, only performance related dimensions are taken into account and integrating other dimensions into the decision process are left as future work.

3.4.2 Security and privacy issues

Increased sensitivity of end users to product security and privacy will determine their choice of the aforementioned cloud deployment models: public, private or community. The most sensitive users with top-secret products to analyze will deploy private clouds behind firewalls. No information should leak outside unless their security policies are breached from outside or inside. However, these users will potentially trade-off cost savings and compromise high utilizations.

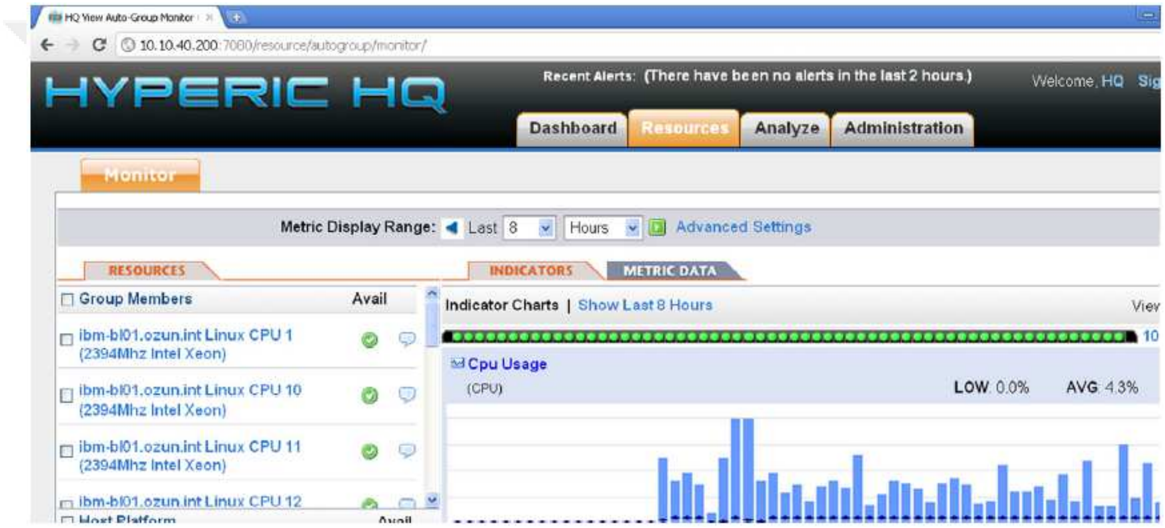
FEA presents a special case in privacy such that most matrix operations require only additive and multiplicative algebra. Fortunately, for a limited set of algebraic operations including addition and multiplication there are homomorphic encryption schemes that allow operators to carry out the necessary calculations over the encrypted data (rather than unencrypted or clear text data) while preserving the correctness of computation as well as the privacy of the original data. This way even a potential malicious attacker inside the service provider would not be able to reconstruct the design details of the original customer product. Basically, after these transformations one cannot differentiate whether the matrix belongs to the chassis of a stealth plane or a toilet pump. Briefly, let R and S be two sets and the encryption function be $E: \ R \rightarrow \ S$. Additively homomorphic means: $E(a + b) = PLUS(E(a), E(b))$ and multiplicatively homomorphic means: $E(a \times b) = MULT(E(a), E(b))$ where E is given by the function $y = E(x) = a \ mod \ n$, where $a = x + r \times p$ and the decryption is given by $x = D(y) = y \ mod \ p$. Homomorphic encryption technique with basic matrices is tested and found that vector operations (additions and multiplications) can be successfully completed over encrypted data, which is the basis for FEA. However, this strategy is not integrated into CalculiX and SPOOLES tools, yet. Homomorphic encryption has also been used in the database field for running privacy-preserving queries over data stored at cloud service providers and in wireless sensor networks for privacy-preserving data aggregation. Therefore, the same techniques could be applied for privacy-preserving FEA in the cloud as well. This topic requires further investigation and constitutes the future work.

Publicly deployed cloud services are potentially prone to both insider (i.e. service provider) and outsider (i.e. man-in-the-middle) attacks. Using the well-known public and private key encryption methods (e.g. SSL used in HTTPS) as well as applying Message Authentication Codes (MAC) over the exchanged FEM models can alleviate some of these problems, namely the data security and integrity can be protected. However, these security precautions should not hinder sharing of data among cloud users. One way to allow controlled sharing is to use Authorization-Based Access Control (ABAC) methods [61] instead of Role-Based Access Control (RBAC), where the former allows finer-granularity control over the resources. In ABAC, users can assign certain usage rules or quotas (e.g. using SAML certificates) over server, storage, and network resources or over data stored in file systems and delegate these privileges to other users. A privilege works for only a given resource and provides limited access. RBAC (e.g. username/password and group) allows unlimited access to all resources owned by a user/group. Therefore, even a small inadvertent mistake done at the user level or at a higher privilege level (e.g. root or admin) can result in the exposition of significant amount of personal or organizational data.

3.4.3 Pricing and accounting

Cloud service providers and telecommunication companies have adopted several models for pricing and charging [81], [82], [83]. These include monthly fixed charging (for infinite or quota-based use) or pay-per-use (the "use" can be per analysis, a batch of analyses, or per CPU/h). In both cases the user pays at the end. This way the users are saved from making large risky investments up front, also known as the Capital Expense or CapEx. Customers only incur the costs of what they use, or the Operational Expense or OpEx, and they may decide to change their business model or IT scaling during this trial period reducing the IT risk involved with the CapEx. However, there are certain scenarios where bulk purchasing becomes a better option and service providers should also consider this alternative in their pricing strategies: e.g. some organizations have quarterly or annual budgets and if they cannot spend the budget allocated for their unit, then they lose the option to use it even in the future budget cycles. In those cases, an option such as buying "10,000 FEA" or "10,000

CPU/h worth of processing power for FEA" becomes an alternative. To be able to charge the users based on different accounting models, there is a need to monitor the server CPU and memory usage or analysis counts accurately. Currently, the Hyperic HQ tool shown in Figure 15 and Hyperic Sigar API is being tested for online per-process resource (CPU, memory, network) monitoring. In this scenario, there is also additional counters attached to the scheduler, which tracks the jobs completed per client.


Recent Alerts: (There have been no alerts in the last 2 hours.)

Metric Display Range: Last 8 Hours

Group Members	Avail
ibm-bl01.ozun.int Linux CPU 1 (2394Mhz Intel Xeon)	
ibm-bl01.ozun.int Linux CPU 10 (2394Mhz Intel Xeon)	
ibm-bl01.ozun.int Linux CPU 11 (2394Mhz Intel Xeon)	
ibm-bl01.ozun.int Linux CPU 12	

Cpu Usage (CPU) LOW: 0.0% AVG: 4.3%

Figure 15: Overall (or per process) CPU, memory network and other resource usages can be tracked with Hyperic for cloud service accounting.

3.4.4 Standardization and portability

Strong progress on horizontal integration of cloud services has been accomplished and there is ongoing work on vertical integration. Standards such as Open Virtualization Format (OVF) allow IaaS providers to export and import Virtual Machines, which could be created at other provider sites. This way the customers can move their data and tasks freely based on cost, performance, or usability criteria without worrying about vendor lock-ins. OVF is currently supported by many hypervisors underlying the IaaS. Other standardization bodies such as Open Grid Forum's Open Cloud Computing Interface (OCCI) [63] also focused their initial efforts of IaaS interoperability, but over time evolved to include PaaS and SaaS layer integration issues. The FEA service can currently import several of the standard input FEM formats and exports results in the FRD file format that can be visualized using CGX. In the future, the list of supported input, output file formats will be expanded so that users can take away their analysis results easily; there are various open-source conversion tools that can be used for this purpose.

Due to the government compliance and security privacy issues people in different countries may be banned from using international FEA services (e.g. consider top-secret military designs). Therefore, replicating and provisioning cloud FEA services in different parts of the world may still be a valuable proposition.

CHAPTER IV

EVALUATION AND CHARACTERIZATION OF PLATFORM AND INFRASTRUCTURE LAYERS

In previous chapter, only the direct solvers included in the SPOOLES [71] library were used. In this chapter, a common framework for comparison of direct vs. iterative solvers is provided; and PETSc library [84] is chosen for this purpose. Iterative methods are favored over direct methods when the memory-space requirements of the matrices are beyond the capabilities of the computing system in hand (either standalone or distributed). However, direct solvers can also be favored when various jobs are executed in an HPC environment and stability as well as the precision of the solutions are of critical importance. Iterative solvers may be more effective and accurate on particular domain-related situations. Hence, a fair comparison is necessary.

The sparse, multi-frontal direct solver MUMPS [85] is also used. MUMPS is the main public domain multi-frontal solver that can be easily integrated into PETSc. Another reason for using MUMPS instead of SPOOLES is that, this multi-frontal solver is also ideal for parallelization. The performance of MUMPS library with Cholesky and LU decompositions of the generated matrices is evaluated. In order to evaluate the performance of iterative solvers, various Krylov subspace methods [86] (CG, BiCG, GMRES) combined with representative preconditioners (BJacobi, SOR, ASM) within the PETSc library are used.

4.1 Platform Layer: Design of Automated Parallel Solvers

First, the direct and iterative solvers as well as the preconditioners are briefly described in Subsection 4.1.1 and then the finite element models (FEM) used to test these solvers are explained in Subsection 4.1.2.

4.1.1 Direct and Iterative Solvers

Krylov subspace methods are used to solve linear system of equations by iteratively multiplying the matrix A starting with vector b, resulting in $A.b$, $A^2.b$, and so on, until it converges [87]. Widely-used algorithms of Krylov subspace methods include Conjugate Gradient (CG), Biconjugate Gradient (BiCG), and Generalized Minimum Residual (GMRES). These methods are evaluated first with no preconditioning and then by preconditioning the matrices using Block Jacobi (Bjacobi), Additive Schwarz Method (ASM), and Successive Over Relaxation (SOR) methods. These preconditioning methods are expected to reduce the condition number of the matrix, therefore reducing the number of iterations for convergence, thus leading to faster solutions. Since readers are well-versed with these methods and concepts, the details are skipped here for brevity [88] [89].

In order to develop a comparison environment for iterative and direct methods, the PETSc (Portable, Extensible Toolkit for Scientific Computation) library [84] is selected. In order to evaluate the behavior of direct solvers, MUMPS [90], [85] that can straightforwardly be used within PETSc is chosen. The performance of MUMPS with LU and Cholesky decompositions is assessed. For the iterative solvers, various Krylov Subspace algorithms that are combined with preconditioners (ASM, SOR, BJacobi) all within the PETSc library are used. A well matching solver-preconditioner combination is critical and readers are referred to the related literature to learn about the implementation details of these methods [88], [91], [92], [93], [94].

The performance comparisons of direct and iterative methods are done with respect to memory usage and multi-core and multi-node execution times over an HPC cluster, consisting of 8 x IBM HS22 Blade Servers each with 2 Intel-Xeon CPUs (8 physical cores) and 24 GB memory (up to 4 nodes are used in these analyses).

4.1.2 Finite Element Models

Figures 16, 17, 18 show the three classical linear elasticity problems, namely a Cantilever beam, Lame problem and Stress Concentration Factor (SCF) modeled for comparing different iterative and direct methods. Cantilever beam is one of the simplest problems for testing strength of materials, where one of the ends of a beam is clamped and the other end is subjected to a vertical force that causes bending. Lame is an axial symmetric problem [89] [95] where an annular cylinder is subjected to uniform pressures both from outside and inside. In SCF problem, there is a hole in the object and the outer dimensions go to infinity. The force lines are denser around the hole, which causes stress concentration potentially leading to crack initiation. These models theoretically represent many real-life mechanical situations in structural engineering, namely aerospace, automotive, construction and machinery industries.

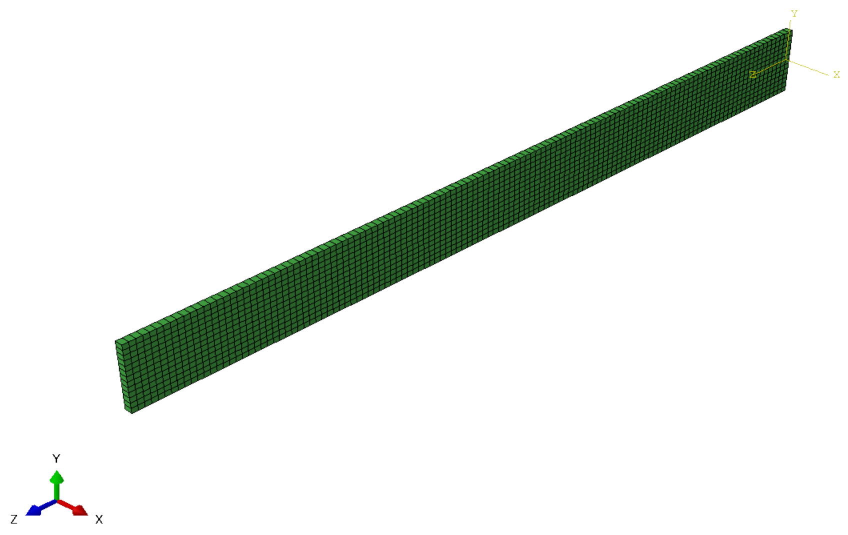


Figure 16: Cantilever beam.

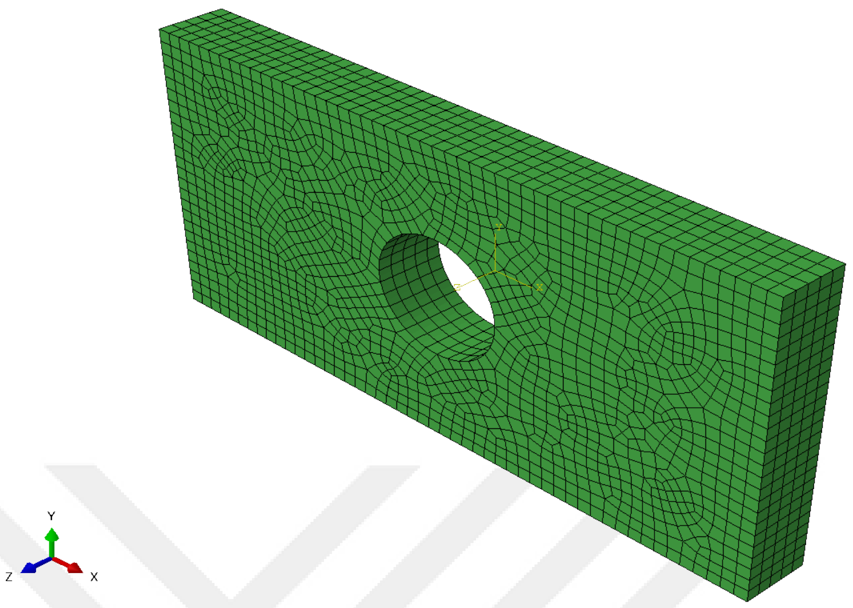


Figure 17: Stress Concentration Factor (SCF).

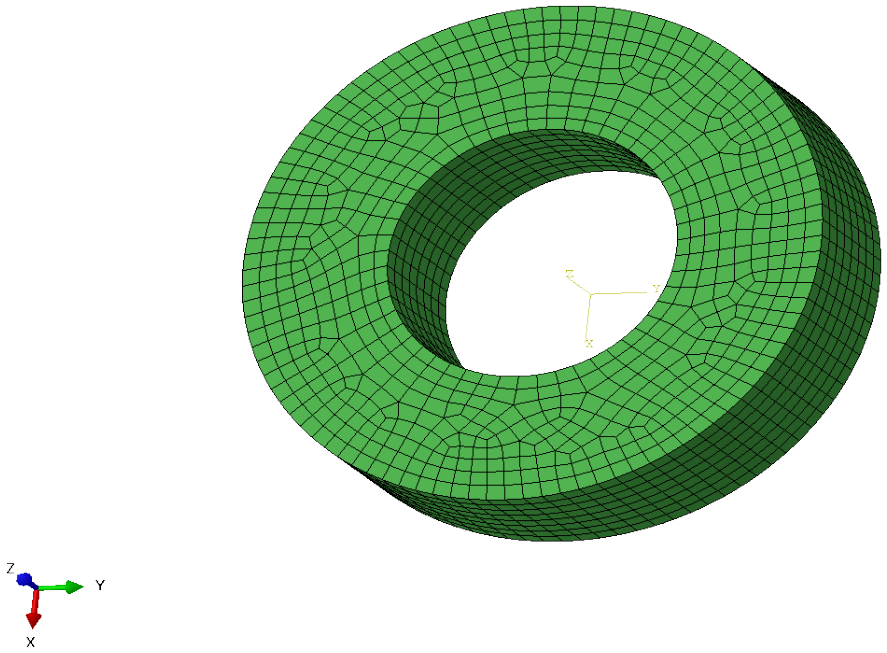


Figure 18: Lame problem.

These three linear problems are meshed with increasing granularities using the CUBIT tool [96] to create their finite element models. As shown in Table 8, there is an inverse correlation between the number of elements and average size of an element; as the mesh gets finer the average size of elements gets smaller. This will end up with more degrees-of-freedom (larger matrices) and larger rank for the solution space (M). This method creates more accurate results but are computationally more expensive. Since elements in a finite element mesh only interact with the surrounding elements and don't have any effect with the remaining ones, the finer meshed problem domains yield sparse matrices. Because of this phenomena, the Non-Zero Elements (NZE) are noted to calculate the degree of sparseness of the resulting matrices that are solved using sparse linear equation solvers.

As shown in Table 8, Cantilever, Lame and SCF problems are meshed in 5 different sizes (minimum, small, medium, large, maximum) and their maximum sizes respectively had approximately 32 million, 739 million, and 956 million NZE. The (NZE) and sparsity ($NZE/(M \times M) \times 100\%$) ratio of the resulting matrices are noted. In addition to these, the resulting matrices are solved using sparse linear equation solvers utilizing either iterative (e.g. CG) or direct (e.g. LU) algorithms. It is previously observed [35] that both the CPU and memory requirements needed to solve a FEM are directly related to the NZE and matrix properties. In this chapter, the possible effects of the condition number of the matrices on iterative solution performance are also examined. Inclusion of effects of other advanced matrix properties such as condition number, wavelength, and bandwidth will be part of the future work.

4.2 Performance Comparison of Solvers

This section summarizes the outcomes for the memory needs and job execution times of fifteen test cases given in Table 8. The performance collations of iterative vs. methods are carried out with respect to multi-core and multi-node execution speeds

Table 8: Matrix properties of Cantilever, Lame and SCF problems for several degrees of mesh coarseness.

Model Name	Job Size	Element Size	Elements	Matrix Size	NZE	Sparsity
Cantilever	Minimum	32	90	768	33840	% 94.26
	Small	16	784	3456	163800	% 98.62
	Medium	8	5464	14040	705240	% 99.64
	Large	4	44992	78312	5136840	% 99.91
	Maximum	2	346921	459918	32637744	% 99.98
SCF	Minimum	80	89	2214	236466	% 95.17
	Small	40	1146	18708	2636748	% 99.24
	Medium	20	6790	98343	15073191	% 99.84
	Large	10	55770	736905	120593655	% 99.97
	Maximum	5	448440	5659596	956996262	% 99.99
Lame	Minimum	256	93	1941	226017	% 94.00
	Small	128	378	12768	1801044	% 98.89
	Medium	64	1625	77487	12051837	% 99.79
	Large	32	18750	587262	96948108	% 99.97
	Maximum	16	125000	4347846	739238994	% 99.99

and as well as RAM usage over an HPC cluster that consists of 8 x IBM HS22 Blade Servers with two Intel-Xeon CPUs(8 physical cores), 24 GB of RAM and Linux operating systems (Red Hat Enterprise) on each of them.

First of all, for all the finite element jobs under consideration, ”minimum”-sized job was too tiny both in terms of CPU-time requirements as well as memory needs. They failed to show the differences either between direct and iterative methods, or preconditioners. Almost all of these executions were completed under one second and therefore parallelization attempts only made the run times worse as job decomposition needs overshadow parallelization’s benefits. Therefore, minimum and small tasks ($NZE < \sim 1,000,000$) are not suitable for HPC. The examinations will be continued with outcomes for medium and larger-sized ($NZE > 1,000,000$) models with and without preconditioners.

4.2.1 Results without Preconditioning

Figure 19 shows performances of solvers with the medium-sized SCF, Lame, and Cantilever jobs having $(1,000,000 < NZE < 10,000,000)$. These jobs benefit in time domain from increased parallelization until 16 core count after which the gains diminish; the small Cantilever problem is an exception since it can have decreasing or varying performance, especially with the Iterative solvers, for increased parallelization. Direct algorithms (LU and Cholesky) can perform better than iterative algorithms (CG, BiCG) only for the Cantilever case, whereas in the larger Stress Concentration Factor and Lame problems these algorithms demonstrate better outcomes up to $\sim 3x$ faster (e.g. collate Direct-LU vs. Iterative-CG). To summarize, in addition to a possible benefit of $2x - 3x$ speedups, an additional $2x - 3x$ speedup can be gained when an appropriate solver is used. Direct algorithms should be preferred for smaller jobs and iterative ones are more feasible for the larger models. Finally, there is a need to stress that the results demonstrating the performance observations of iterative algorithms includes no preconditioner effects. In Figure 19d, it can be noticed that iterative algorithms use less memory than direct ones and among the direct algorithms, Cholesky uses less memory than LU. Since more memory allocation also means slower execution speeds, time measurements observations also agree with the corresponding memory consumption.

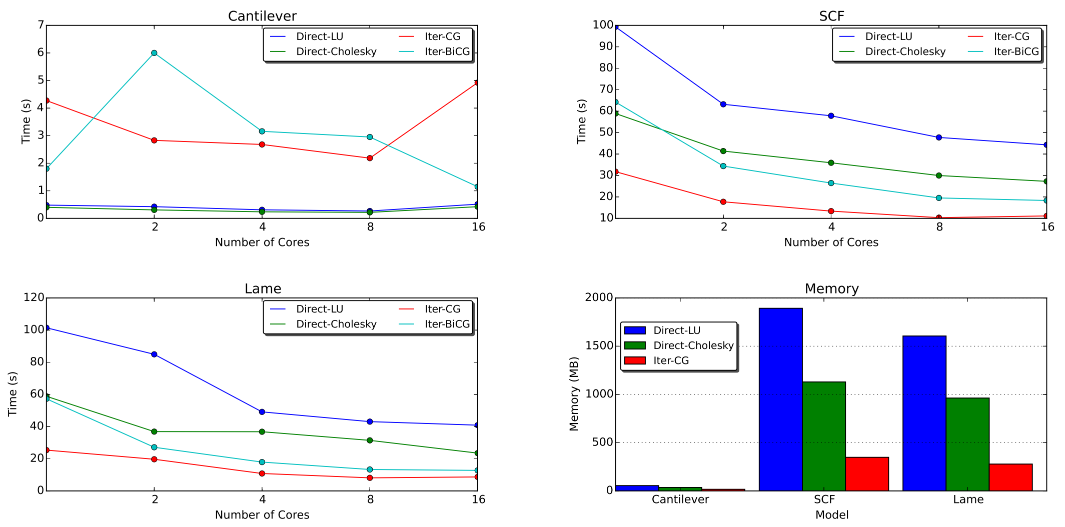


Figure 19: Execution Time (sec) and Memory (MB) usage of medium-sized Cantilever, SCF and Lame jobs.

Table 9: Jobs ordered in increasing number of NZE. Performance of best direct and iterative solvers compared.

Name		Size	NZE	1-Core Time (s)	Fast. Time (s)	Core N. (Fastest)	Time (s)	Core N. (Fastest)
Cantilever	Min	768	33480	0.02	0.02	1	0.02	1
Cantilever	Small	3456	163800	0.19	0.19	1	0.071	1
Lame	Min	1941	226017	0.06	0.06	1	0.095	1
SCF	Min	2214	236466	0.09	0.08	4	0.084	1
Cantilever	Medium	14040	705240	2.00	0.98	8	0.217	8
Lame	Small	12768	1801044	1.22	0.57	8	1.272	8
SCF	Small	18708	2636748	0.93	0.48	8	1.857	8
Cantilever	Large	78312	5136840	12.38	4.83	8	4.206	4
Lame	Medium	77487	12051837	13.14	4.21	8	23.51	16
SCF	Medium	98343	15073191	13.97	6.52	8	27.264	16
Cantilever	Max	459918	32637744	139.90	47.26	16	71.37	8
Lame	Large	587262	96948108	169.07	44.87	16	644.14	32
SCF	Large	736905	120593655	159.11	47.96	16	703.96	32
Lame	Max	4347846	739238994	2924.25	443.42	32	SWAP	SWAP
SCF	Max	5650596	956996262	2823.73	550.21	32	SWAP	SWAP

Results in Figure 20 and Figure 21 for the Large and Maximum sized jobs generally provided matching performance figures and outcomes. Within the test suite, only Lame-max and SCF-max tasks gained from parallelization using 32 cores. Figure 21 shows the run times of iterative algorithms without preconditioners for these maximum-sized tasks. Direct algorithms could not return the results for the two maximum sized jobs for SCF and Lame having $\sim 740,000,000$ and $\sim 960,000,000$ NZE, respectively. Their memory consumption hit the maximum capacity of nodes approximately around 22 GB; and the virtual memory management function of the operating system triggered the utilization of disk swap areas in order to satisfy the memory needs. Those results are not recorded and reported because of the non-reliable nature of the disk swap area usage. Here, the conclusion is that the iterative algorithms not only improve the performance over direct ones, but also make it feasible for huge tasks ($NZE > 500,000,000$) to properly return a result where direct algorithms completely fail. Figure 19 demonstrates the run times of iterative algorithms without using preconditions.

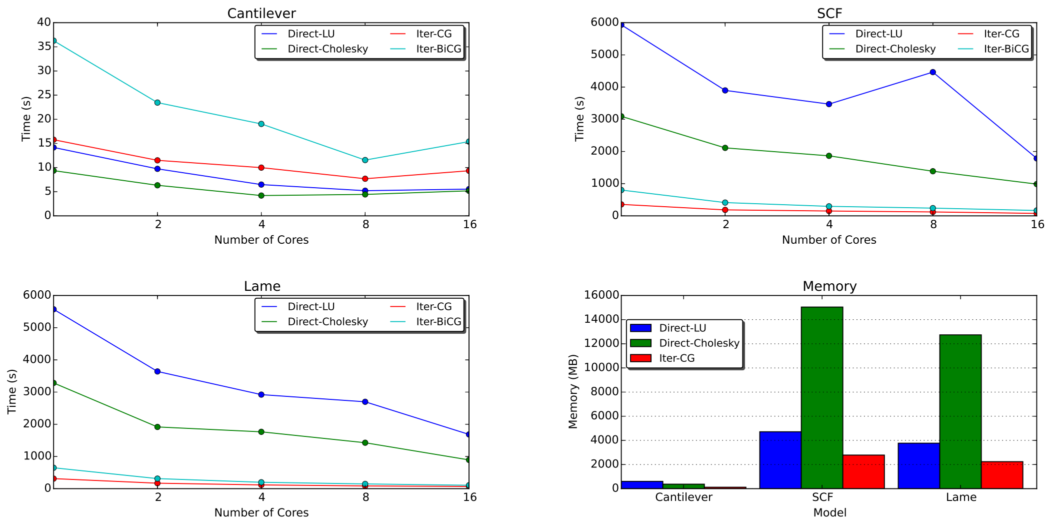


Figure 20: Execution Time (sec) and Memory (MB) usage of large-sized Cantilever, SCF and Lame jobs.

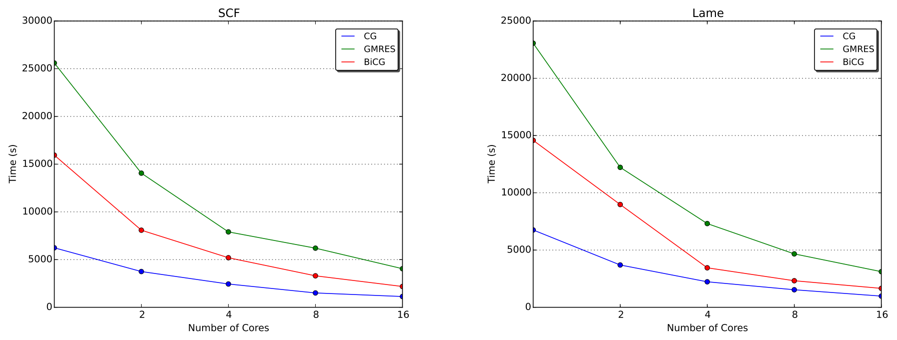


Figure 21: Execution Time (sec) of max-sized SCF and Lame jobs.

4.2.2 Results with Preconditioning

In this subsection, the effect of preconditioners are examined to figure out the performance trade-offs. It is noticed that preconditioners provide notable gains in execution speeds, but at the same time it can also increase memory consumption compared to tasks that are run without preconditioners. All the execution times published in this subsection include the time that is necessary for the application of the preconditioner. Here, it is also of special importance to contain the preconditioning time needed, because this total time gives a fair premise for collation with no-preconditioner cases. Note that the preconditioning time converges to a negligible portion of the total run time when solution times get bigger.

Figure 22 shows that effects of preconditiners, in general, improves execution speeds of iterative algorithms (CG, BiCG, GMRES) except for the cases of SOR with BiCG and ASM preconditioner with CG. The combination of CG iterative algorithm with Bjacobi preconditioner with its 47.6 second run time, has a 25% faster performance than the combination of CG-SOR (63.5 sec) and 36% faster performance than GMRES-BJacobi combination(74.8 sec), which are the second and third best performing combinations after CG-Bjacobi. To summarize, preconditioning can enhance execution speed by $25 - 36\%$.

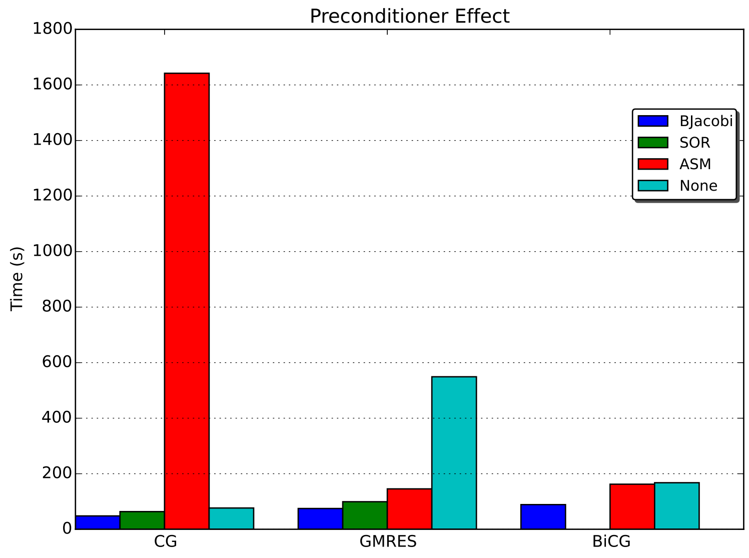


Figure 22: Effect of different preconditioners on execution times of different Iterative solvers.

4.2.2.1 Preconditioning Effect on Memory

Figure 23 demonstrates the memory consumption of the Cantilever task for different degrees of mesh coarseness with the Direct (LU, Cholesky) algorithms and Iterative-BiCG algorithm that uses no preconditioning, a Block-jacobi preconditioned matrix and an ASM preconditoner. As expected, the memory need increases as NZE grows and the log-log figure indicates an average of 30-100 bytes per element (e.g. 30MB-100MB/1 Million NZE). Direct algorithms use more memory than iterative ones and between the direct algorithms LU option requires more memory than the Cholesky option. While Block-jacobi preconditioner does not have a big impact on memory consumption, on contrary ASM can remarkably increase the memory consumption (by 50-100%). A very similar impression for the CG and GMRES algorithms is acquired, therefore their combinations with BiCG are excluded for briefness. One significant point is that the SOR preconditioner does not function together with BiCG. Also, the outcomes for the Lame and SCF tasks and their memory consumption patterns for various degrees of mesh coarseness have exhibited the same structure, where the direct algorithms demand more memory than the iterative ones.

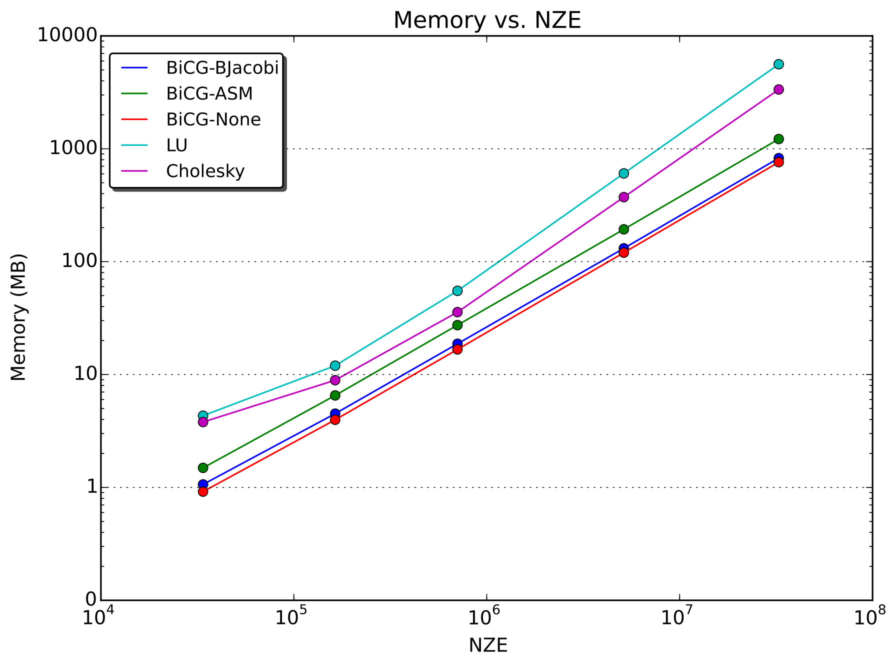


Figure 23: Memory usage of direct-vs-iterative BiCG method with different preconditioning; for Cantilever problem at different granularities (NZE).

4.2.2.2 Preconditioning Effect on Execution Time

Figure 22 shows that Preconditioning in general improves execution times of iterative methods (CG, GMRES, BiCG) except for the ASM preconditioner for CG. The best preconditioner for all Iterative solver types is Block-Jacobi (Bjacobi).

In Figure 24, the preconditioner is set as Bjacobi and compare the time performances of different Iterative methods for the biggest job: SCF problem meshed at maximum granularity. The performance between the best Iterative solver CG, and second best BiCG is $\sim 2x$ and between CG and GMRES it is $\sim 4x$. To summarize, iterative solver Conjugate Gradient (CG) with the Block-Jacobi preconditioner demonstrates the most outstanding performance among the test cases.

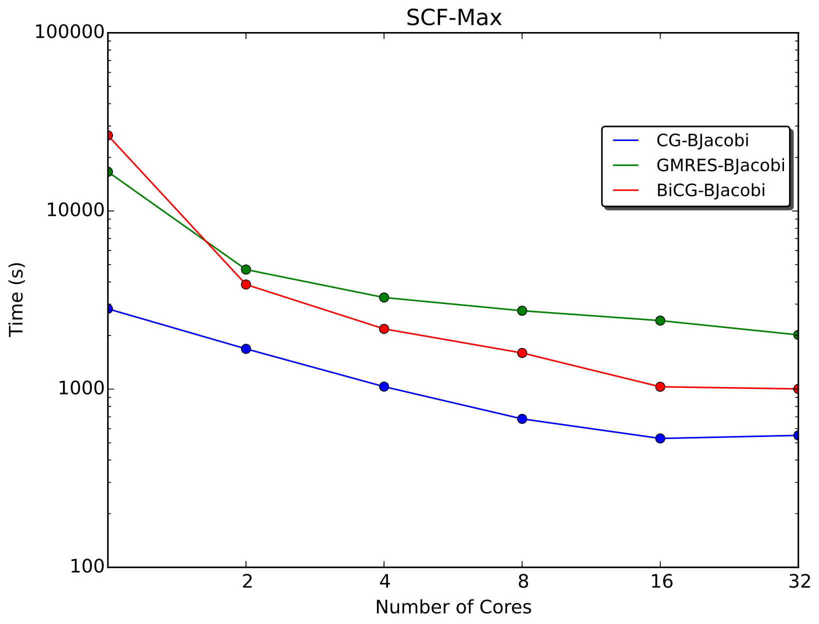


Figure 24: Comparison of solvers with a fixed preconditioner.

4.2.3 Comparison of the Best Direct and Iterative Methods

Table 9 compares the performances of best direct and iterative methods. Until now, the outcomes show that the Cholesky algorithm is the best performing direct solver meanwhile Conjugate Gradient algorithm with a Block Jacobi preconditioner is the best iterative-preconditioner choice. Next, the run time characteristics of these iterative vs. direct algorithms are collated and describe the outcomes for all of the tasks in Table 10. It can be rapidly pointed out that direct algorithms including Cholesky method cannot successfully return results for the maximum-sized Lame and SCF models having $\sim 740,000,000$ to $1,000,000,000$ NZE. However, the Iterative CG-Bjacobi combination completes these jobs within $7-10$ minutes time range. As the job sizes increase, they also benefit more from parallelization. However, if there is enough memory and time direct solvers are guaranteed to return a result, whereas the Iterative solvers may not converge even if there is enough memory when they encounter ill-conditioned matrices. These issues will be discussed in Section 4.2.4.

Table 10: Non-Zero Element ordering for Cantilever (C), Lame (L) and Stress Concentration Factor (S) Jobs including a performance collation of best iterative and direct solutions.

Job Properties			Iterative-CG-BJacobi			Direct-Cholesky	
Name	Size	NZE	Time (ms)1-core	Time (ms) Fastest	Core # Fastest	Time (ms)	Core # Fastest
C-min	768	33,840	0.02	0.02	1	0.02	1
C-small	3,658	163,800	0.19	0.19	1	0.071	1
L-min	1,941	226,017	0.06	0.06	1	0.095	1
S-min	2,214	236,466	0.09	0.08	4	0.84	1
C-medium	14,040	705,240	2.00	0.98	8	0.217	8
L-small	12,768	1,801,044	1.22	0.57	8	1.272	8
S-small	18,708	2,636,748	0.93	0.48	8	1.857	8
C-large	78,312	5,136,840	12.38	4.38	8	4.206	4
L-medium	77,487	12,051,837	13.14	4.21	8	23.51	16
S-medium	98,343	15,073,191	13.97	6.52	8	27.26	16
C-max	459,918	32,637,744	139.90	47.26	16	71.37	8
L-large	587,262	96,948,108	169.07	44.87	16	644.14	32
S-large	736,905	120,593,655	159.11	47.96	16	703.96	32
L-max	4,347,846	739,238,994	2,924.25	443.42	32	SWAP	SWAP
S-max	5,650,596	956,996,262	2,823.73	550.21	32	SWAP	SWAP

Although HPC clusters are used, the memory management modules of the local nodes and their OS can create problems for HPC jobs; i.e. they are not optimized for managing large chunks of memory beyond 8-10 GBs, a modern OS (Linux in this case) installed in the 64-bit machines with 24GB physical memory can certainly allocate all available memory to processes, but this allocation and de-allocation has significant diverse impact on execution times [97]. It is observed that most of the time is spent in allocating or moving the memory rather than solving the matrices. Operating System enhancements such as permitting HPC user processes and related data structures to benefit from the slab allocator can be fairly helpful.

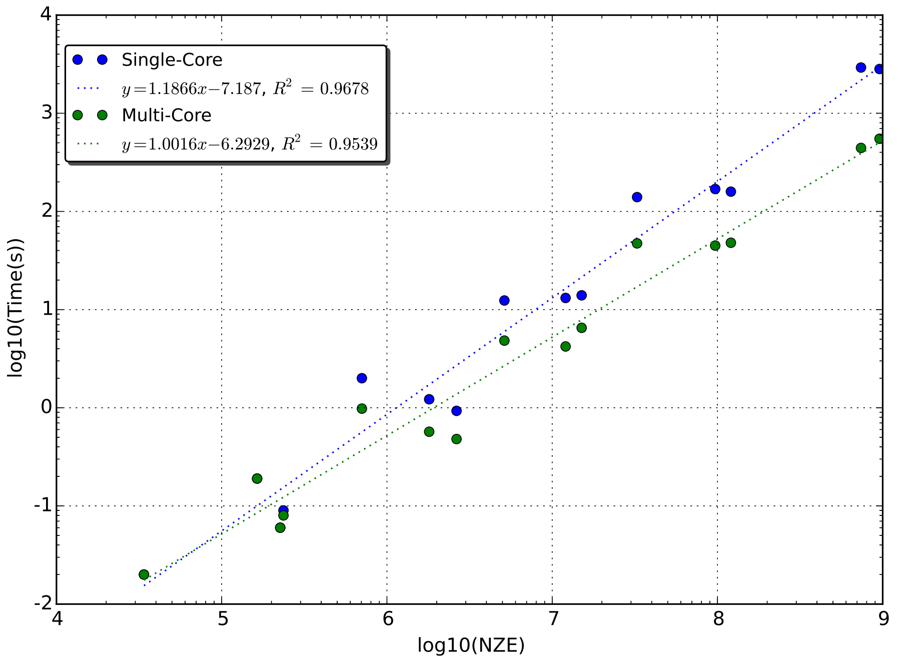


Figure 25: A first-degree polynomial fit for prediction of execution times using NZE of matrices.

Figure 25 demonstrates linear polynomial fit for NZE vs. execution time on a logarithmic scale for single and multi core cases. For the single-core case the identified linear function is:

$$\log_{10}(time) = \log_{10}(NZE) \times 1.1866 - 7.187 \tag{1}$$

and for the multi-core case, the identified function is:

$$\log_{10}(time) = \log_{10}(NZE) \times 1.0016 - 6.293 \tag{2}$$

The related R^2 values of 0.9539-0.9678 are close to 1, indicating a fine fit. In the next subsection, an exploration on the fact that divergence from the regression line happens in situations where the matrix under consideration is ill-conditioned to an extent, is given.

4.2.4 Effect of Condition Number on Performance

The condition number (CN) for a nonsingular matrix A given by Equation 3:

$$K(A) = \|A\| \, . \, \left\|A^{-1}\right\| \tag{3}$$

can be utilized to decide how quickly an iterative algorithm may converge to a solution [98]. For ill-conditioned cases where $K(A)$ is large, solution speed (thus convergance) can be quite slow-going or may even diverge. Figures 26, 27, 28 exhibit that as the coarseness of the meshes for SCF, Cantilever and Lame models decrease, their corresponding matrices will become more diagonal, which also contributes to the reduction of their condition numbers as demonstrated in Table 11.

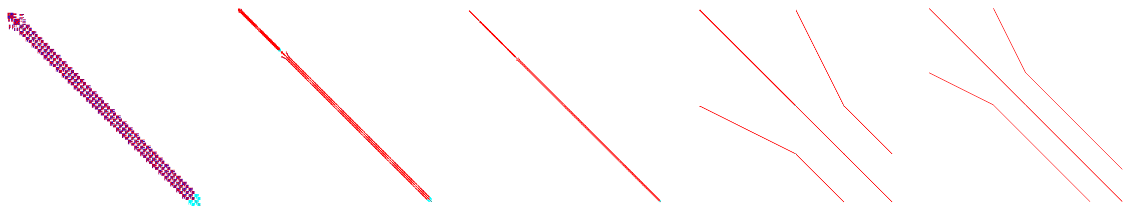


Figure 26: Matrix bandwidth properties of Cantilever Problem, (a) Min, (b) Small, (c) Medium, (d) Large, (e) Max. Matrices tend to have a more diagonal structure as the mesh gets finer.

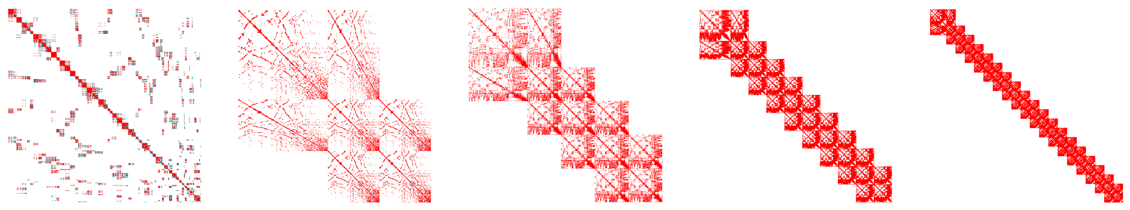


Figure 27: Matrix bandwidth properties of SCF Problem, (a) Min, (b) Small, (c) Medium, (d) Large, (e) Max. Matrices tend to have a more diagonal structure as the mesh gets finer.

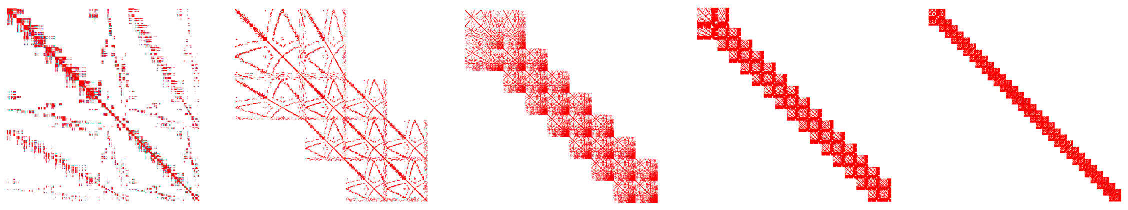


Figure 28: Matrix bandwidth properties of Lame Problem, (a) Min, (b) Small, (c) Medium, (d) Large, (e) Max. Matrices tend to have a more diagonal structure as the mesh gets finer.

The condition numbers, K, are calculated by taking the ratio of maximum eigenvalue to the minimum using PETSc library. The results are given in Table 4. A careful examination of the K (Max. CN) values shown in Table 11 in parallel with the performance outcomes demonstrated in Table 10 tells that using the K values together with the NZE of two meshed objects can help to describe their performance dissimilarities. For example, while the SCF-small task shown in Table 10 has around 30% more NZE than Lame-Small task, it executes and returns a result in about 25% shorter time (0.93 vs. 1.22 sec with 1 core and 0.47 vs. 0.57 when 8 cores are used). Another almost identical deviation is encountered among Lame-SCF Large tasks. This outcome can be related to the larger condition number of Lame-Small task (50.29×10^{13}) collated to the condition number of SCF-small task (13.35×10^{13}). One challenge faced when the condition number is included as a parameter into FEA execution time prediction efforts, is that the condition number computation itself may not converge. Figure 29 shows one such case, where the condition number computation for one of the well-conditioned matrices converges quickly, whereas the ill-conditioned one oscillates for thousands of iterations without returning a result. Since the performance variabilities due to condition number are relatively minor ($< 10\%$) in these cases, its comprehensive study and incorporation in the equations are left as future work.

Table 11: Convergence behavior of Condition Number computations of the matrices from test suite.

Job Name		K (Max CN) $C \times 10^{13}$
Cantilever	Minimum	5.384
	Small	5.384
	Medium	5.384
	Large	0.462
	Maximum	0.216
SCF	Minimum	19.64
	Small	13.35
	Medium	8.19
	Large	4.40
	Maximum	2.39
Lame	Minimum	66.98
	Small	50.29
	Medium	28.32
	Large	14.40
	Maximum	7.06

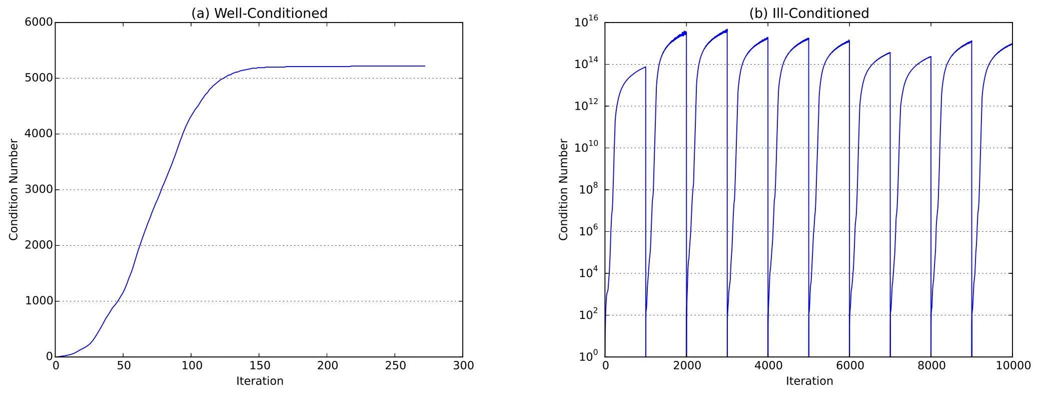


Figure 29: Computation of condition number for well vs. ill conditions.

4.3 Infrastructure Layer: App Containers vs. Physical and Virtual Machines

A typical HPC-as-a-Service supplier using containers need to supply service for multiple, geographically scattered request with various demands. Docker is a recently developed technology being used in Cloud Computing environments as a solution to overcome the dependency problems of jobs being submitted to the infrastructure. So that the cloud computing paradigm aims to be as generic as possible, there is an ever increasing task of handling and coordinating multiple libraries, software packages and application-specific environment variables for service suppliers. By using container technology, this issue may be handled by using predefined instances with specific software and hardware characteristics encoded and deployed easily whenever needed. This approach also guarantees the existence of the resources as requested and encapsulates the performance metrics so that it can't be affected by other jobs running concurrently within the same system. This isolation within the cluster, makes different applications with different dependencies co-exist, run and utilize the underlying hardware without conflicts. Another advantage that can be mentioned about utilization of containers is that they do not run on hypervisors like virtual machines but run on operating system's kernel, share the kernel and standard application libraries of the operating system. This property of containers makes them lightweight so that they can be distributed and loaded quickly whenever a need uprises and exhibit similar performance characteristics as bare-metal applications.

Figure 30 illustrates different alternatives for setup and execution of HPC software; in this case, OpenFOAM Computational Fluid Dynamics (CFD) toolkit. In a classic installation scenario, shown as CASE 1, a physical server would be acquired either by buying it or renting it from a cloud data center. Next, a host Operating System (OS) is installed on this hardware and install OpenFOAM over this OS (e.g. by running sudo apt-get install openfoam command in Ubuntu). In CASE 2, a machine with OS will be used and Docker [49] will be installed on top of the Host OS first (e.g. apt-get install docker) as a virtualization layer and use Docker to pull the Ubuntu:latest + OpenFOAM images to prepare the HPC environment. Images on the same physical machine or different machines can communicate via MPI (Message Passing Interface) as usual. In CASE 3, the OS is virtualized by a Type1 (Native) Virtual Machine

Monitoring (VMM) software and OpenFOAM is installed inside that VM. This case provides isolation of all application layer software including HPC at the OS kernel level. VMWare ESX Server [99], Microsoft Hyper-V [100], Kernel Virtual Machine (KVM) [101], and Xen [102] are among the widely used system software for this purpose. In CASE 4, there is excessive layering of virtualization starting with a host OS, a Type2 (Hosted) VMM, and then a guest OS with OpenFOAM inside. This case may only viable for development purposes.

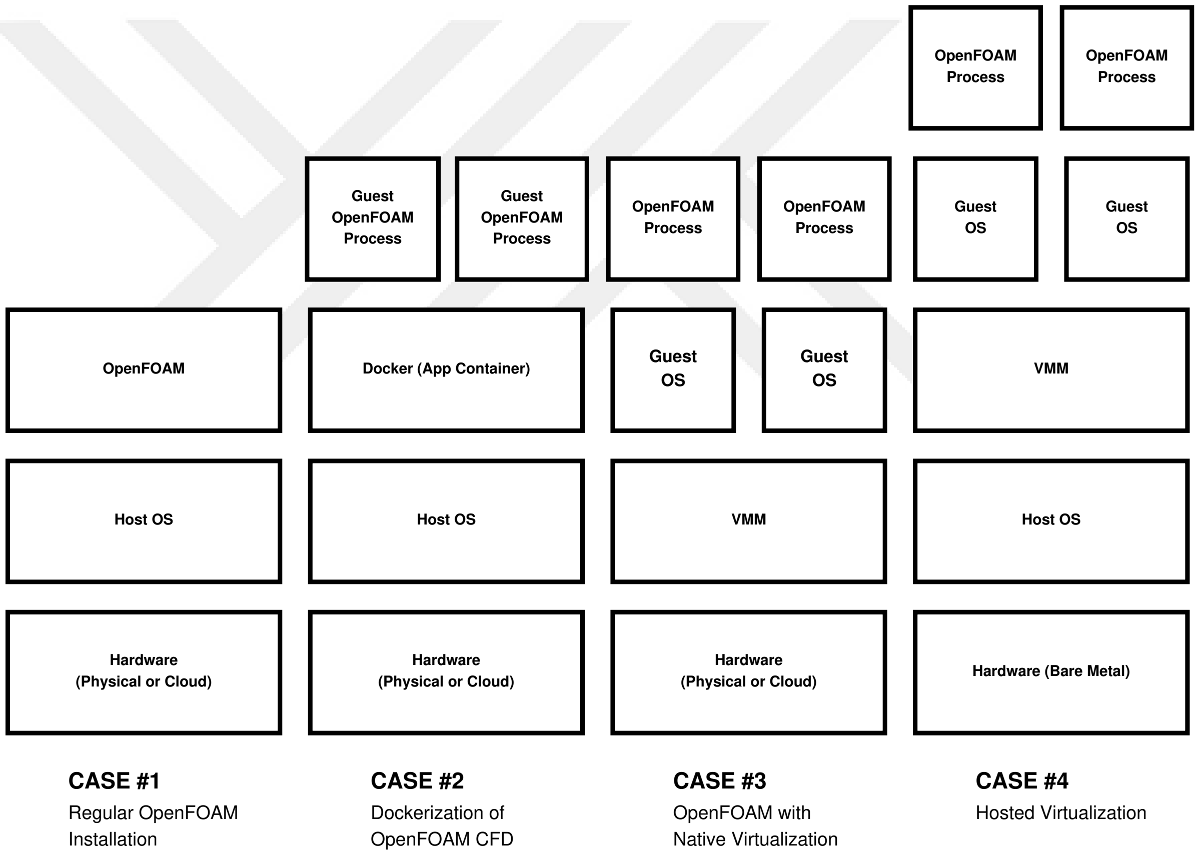


Figure 30: Alternatives for installation of OpenFOAM CFD software over physical, virtualized and dockerized resources.

4.3.1 Application Containers

Process Containers were first implemented and integrated into the Linux Kernel by Paul Menage, et al. [44] starting with version 2.6.x. The goal was to provide resource isolation (CPU, memory, I/O, network) and prioritization among controlled group of processes, called cgroups. Light-weight containers can be setup, configured, shipped, copied, deployed, and terminated much faster than VM clusters, since they run on the same OS kernel and share this kernel's resources, drivers, and libraries. They are also designed to isolate jobs within a cluster; but this is a claim to be tested extensively. Isolation within the cluster could make different HPC and high-throughput applications co-exist and utilize the underlying hardware resources without any conflicts. Finally, containers can remove the need for queue scheduling and resource management software such as PBS/Torque, SLURM, LSF, and YARN, since there is a separate and isolated process for every HPC job to be executed.

Container technologies can be classified into two as ”application containers” and ”system containers” [47]. Docker is an example for the former, whereas Linux Containers (LXC) [48] and OpenVZ for the latter. App containers execute a single process, whereas system containers have full OS stack. Due to its light-weight approach and ease of deployment, Docker [49] is the most widely adopted technology among the systems community for now. It is available for Linux, Windows, and Unix-based OS such as MacOS. Scientists can easily share pre-packaged images that are loaded with scientific software and data via Docker Hub (hub.docker.com). Images can be layered (using a layered file system) to share functionality and save disk space. However, sharing can create resource contention and reduce performance if not handled properly.

4.3.1.1 Docker

Each Docker container is a process that runs in its own namespace for PID, UTS, IPC, Mount, and Network [103]. The storage and file system layers are implemented by union of mount points, a.k.a Unification File System (UFS; e.g. AUFS). Table 12 lists some of the commonly used commands for Docker image creation and management.

You can install software into images by “pulling” layers on top of each other, usually stating with the OS layer, then possibly pulling a database layer and/or a HPC software such as OpenFOAM. You can rename your images and save them into Docker Hub (hub.docker.com/) cloud repository for later use and sharing. Commands in Table 12 help you login, logout, list and manage these custom Docker images. The details are skipped here for brevity, but Table 12 serves as a good reference.

Table 12: Commonly used Docker management commands.

docker pull ubuntu:latest	Install the latest Ubuntu OS on the image
docker run -ti ubuntu:latest /bin/bash	Run docker image interactively (via shell). Ctrl+D exits container, but changes are lost if not committed.
docker run ubuntu:latest echo ”Hello World.” cd ˜ ; mkdir pareng	“Hello World” application for Docker. Make a directory in the image.
docker images	List current docker images
docker rmi image_id	Delete the specified docker image
docker pull postgres:9.4.5	Install a Postgres DB on the image
docker run -d –name postgres_ container postgres:9.4 docker exec -ti postgres_container /bin/bash	Start the postgres loaded image and go the the SQL CLI for postgres
docker ps –l	List running docker images
echo ”FROM ubuntu:latest” > Dockerfile docker build -t new-image:1.0 .	Build new images from existing images using Dockerfile
docker commit dockerid my/image	Save changes
docker login/logout	Docker repository Login and Logout
docker tag my/changedimage myid/webserver docker push myid/webserver	Push docker image to docker cloud
docker port dockerid	Show all mapped ports for dockerid
docker cp	Copy files between container and local file system

4.3.2 CPU-Intensive and I/O Intensive Benchmarks

OpenFOAM Computational Fluid Dynamics (CFD) software can solve an extensive range of fluid flow, heat transfer, mechanical, acoustics, and electromagnetic problems [37]. Parallel simulations are first executed on OpenFOAM for two laminar flow experiments: depthCharge2D and depthCharge3D. These examples simulate a gas bubble exploding inside a water-filled container; which also has air on top the water. The 0.1-0.5-1 seconds of this bubble explosion event is simulated, which indeed takes much longer time to simulate. OpenFOAM uses the compressibleInterFoam solver, which is a solver for 2 compressible, laminar and turbulant, immiscible fluids (iso-thermal or non-isothermal). Simulation files for depthCharge2D from OpenFoam examples are copied into the image, setup the configuration for domain decomposition methods and run OpenFOAM in parallel.

4.3.3 Setting up OpenFOAM on Docker

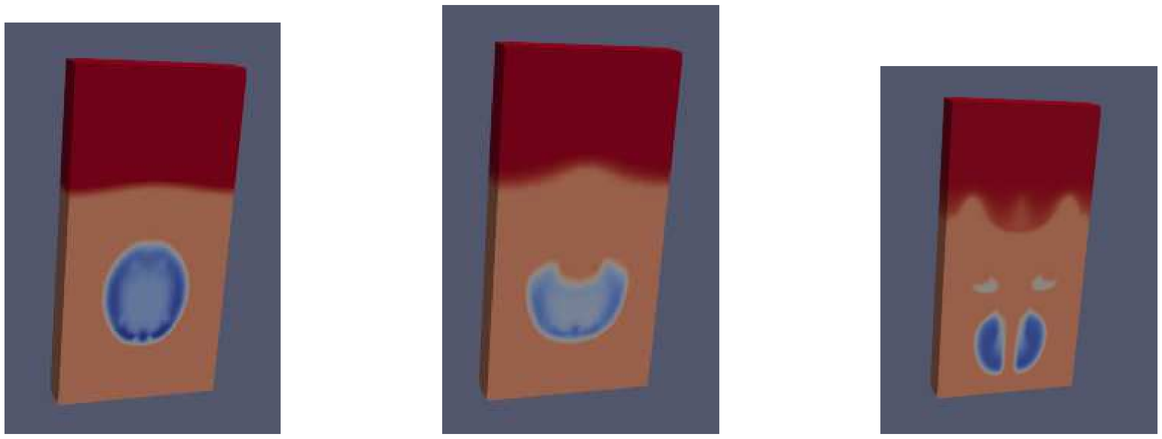
Containers provide similar portability and faster deployment benefits to VMs, except that they are lighter-weight, as they do not replicate the OS-related kernel and driver codes. In that regard, containers can be compared to cloud Platform Services (PaaS), whereas VMs are the infrastructure (IaaS).

4.3.3.1 OpenFOAM

An open-source software called OpenFOAM Computational Fluid Dynamics (CFD) that can solve an extensive engineering range of fluid flow, heat transfer, mechanical, acoustics, and electromagnetic problems [37] is used. Parallel simulations on OpenFOAM for two laminar flow experiments: depthCharge2D and depthCharge3D (see Figure 31) are executed. These examples simulate a gas bubble exploding inside a water-filled container; which also has air on top the water. The 0.1-0.5-1 seconds of this bubble explosion event is simulated, which indeed takes much longer time to compute. OpenFOAM uses the compressibleInterFoam solver, which is a solver for 2 compressible, laminar and turbulant, immiscible fluids (iso-thermal or non-isothermal).

4.3.3.2 Domain Decomposition in Docker: Simple and Scotch Methods

One of the most effectively and widely used parallel computing approaches used during solution of the equation systems resulting from discretizations of partial differential equations is the domain decomposition method (DDM) where the data is divided into subdomains and shared among processes. In non-overlapping version of DDM,



(a) (b) (c)

Figure 31: Solution of DepthCharge2D problem visualized using Paraview software at (a) 0.1 sec, (b) 0.5 sec, (c) 1.0 sec.

all subdomains are independent from each other and can be handled as separate computational entities. Independence of subdomains paves the way for efficient parallelization and scaling of finite element computations. Adaptive mesh refinement techniques can also be used flexibly without any need of change in other subdomains. This method has a great potential for implementation in real life engineering and scientific problems and by modularity, flexibility promises ways to optimize and refactor bottlenecks in the implementation without demanding a need for remodeling of the whole system.

During past years, in order to incorporate this method into actively used software efficiently, several libraries have been implemented. They may be divided into two main groups: 1) Libraries for modeling and solving using DDM (e.g. PETSc and MODULEF)[104] and 2) Libraries for decomposing the domain into optimal subdomains in order to guarantee the scaling of the code using multiple processes (e.g. METIS and SCOTCH). Libraries like METIS and SCOTCH [105] aim to decrease boundaries between subdomains in order to minimize the communication need between processes, while simultaneously trying to keep the computation load of each process balanced with others for optimal CPU usage.

OpenFOAM uses a domain decomposition tool called "decomposePar" with simple, hierarchical, scotch, and metis options; next openMPI is used to parallelize the computation. With the simple method, how to decompose the physical problem geometrically such as ($x = 1$, $y = 2$, $z = 1$) for a 2-core calculation parallelized in y direction is manually specified. With the scotch method, no geometric input is provided, as this selection is automated and the "best" option is selected by the method itself. After the simulation ParaView software is used to visualize the results.

4.4 Performance Comparison and Results for Docker vs. VM

Application containers such as Docker form an alternative to Virtual Machines (VM) and provide faster development as well as deployment of HPC services over cloud computing infrastructures[48]. Throughout the recent section that is based on a previous research [106], performance of Docker containers with virtual machines using OpenFOAM [37] software over a public cloud is collated. It is discovered that while Docker offers $\sim 10x$ gains in new HPC cluster setup times, it has no noteworthy performance handicaps compared to virtual machine technologies or even to infrastructures based physical machines. Domain Decomposition Methods (DDM) including Simple manual DDM and Scotch, which automates the selection of the best geometrical partition, are also compared. Since the workloads (e.g. DepthCharge3D) were highly symmetric, different DDM methods could not differentiate in performance.

In this subsection, architectural alternatives, advantages as well as disadvantages of each alternative, are discussed. The three setups on Scaleway cloud that are representative of the cases shown in Figure 30, physical, virtual and dockerized HPC alternatives will be compared in this section. Performance and isolation benchmarking results for Physical, Virtual and Dockerized HPC alternatives over the public Scaleway cloud will be provided. Scaleway C2S physical servers have 4 (x86-64) cores, 8GB RAM, and 2.5Gbps NIC. For physical cluster tests, first Ubuntu 14.0.4 is used as the OS and OpenFOAM is installed directly on the top of OS. For Dockerized HPC, Docker is installed and OpenFOAM is executed inside those, again on top of C2S servers. For Virtual Server tests, Scaleway VC1M with 4 (x86-64) cores is used, 4GB RAM, KVM for virtualization, Ubuntu 14.0.4 and OpenFoam inside.

- **CASE #1** Physical Server: Scaleway C2S is tested with 4 (x86-64) cores, 8GB RAM, 2.5Gbps NIC; installed with Ubuntu 14.0.4 and OpenFOAM inside.
- **CASE #2** Physical + Docker: Scaleway C2S is tested (h/w same as above); installed with Ubuntu, Docker and OpenFOAM inside.
- **CASE #3** Virtual Server: Scaleway VC1M is tested with 4 (x86-64) cores, 4GB RAM, which uses KVM for virtualization; installed with Ubuntu 14.0.4 and OpenFoam inside.
- **CASE #4** CASE#4 alternative of installing Docker inside a VM image is not tested, since this creates unnecessary layering and comparisons.

Before going into performance comparisons it must be denoted that the biggest contribution of Docker by experience is its setup and bootup times. While it can take about 10-15 minutes to download and install OpenFOAM from repository (sudo apt-get install openfoam30) onto a bare metal server or into a VM, this complete operation takes about 1 minute in Docker (docker pull myid/ubuntu14.04_openfoam3.0.1). The boot time for a physical and virtual machine are also on the order of 1-2 minutes, whereas this value is meaured as a few seconds in Docker. Therefore, these are expected to be the major contributions of Docker, if it passes the performance test as detailed below.

The results for 1-2-4 cores as shown in Table 13, 14, 15, 16 are reported. Linux "time" command inside the Docker shell is used to obtain real, user, and system times separately. "Real" reports the wall-clock time, "User" refers to the total CPU (multi-core) time spent in user mode, and “Sys” refers to the total CPU (multi-core) time spent in kernel mode. These breakdowns are reported where needed; otherwise “real” time values are reported. A separate CPU or memory profiler are not installed as these suites can also affect observed time. Experiments are run three times and the average is reported; the variances are less than 1% in most cases. Table 13 shows the DepthCharge2D 0.5 second simulation results with real, user, system time breakdowns for control purposes. Taking the single (1) core time as reference and using Amdahl's Law, from the speedups for measured times of 2-4-8 cores, the parallelizable portion of the code is calculated to be 62%. The system overhead is ¡ 2%.

Table 13: DepthCharge2D – 0.5 second simulation multi-core time (real, user, sys) results.

Cores	Real (sec)	User (sec)	Sys (sec)
1	520.9 (ref.)	517.7	2.8
2	371.7 (x1.40)	668.9	7.5
4	273.3 (x1.90)	867.4	11.6
8	230.9 (x2.26)	1275.8	21.2

Table 14 shows the results and comparison for timing of the same DepthCharge2D 0.5 second task over a physical, dockerized and virtual machine. It is found that, the physical machine results are the best in all cases, as expected. Docker performs 4,5% better than virtual server on a single core, but 3% worse on 2-core with simple decomposition. In most cases Docker and Virtual are no worse than 5% from the Physical installation. In this performance comparison, simple manual decomposition is also found to work better than scotch. However, note that the problem is domain is uniform and symmetric and the number of cores are relatively low (2-4), therefore not leaving scotch lots of alternatives for optimization. Effect of graph partitioning applications such as SCOTCH and METIS [105] becomes more apparent once the domain is geometrically more complex and the meshing structure is non-uniform throughout the region.

Table 14: DepthCharge2D – 0.5 second simulation results.

	1	2 (Simple)	2 (Scotch)	4 (Simple)	4 (Scotch)
	Time (s)	Time (s)	Time (s)	Time (s)	Time (s)
Physical Machine	522.66	359.84	373.61	269.61	283,39
Physical + Docker	524,1	126.8	0.96	15.36	18.33
Virtual Machine	548.8	383.87	391.28	274.38	282.66

About Docker AUFS vs. Volumes: Results show no major difference for CPU-intensive workloads such as OpenFOAM, since the IO layer is not loaded / tested significantly.

Results for DepthCharge2D - 1 second and DepthCharge 3D - 0.1 second simulation results can be found in Tables 15 and 16. The performance of Dockerized HPC is comparable to (and sometimes slightly better than) both Physical and Virtual HPC setups. Therefore, it is concluded here that there is no significant performance difference among Physical, Virtual and Dockerized HPC choices and the real improvements can be achieved within multi core hardware and HPC solvers and software.

In Table 15, the results for a longer simulation (DepthCharge2D - 1 second) are presented together with the iteration counts for the solver. In general, the time it takes to solve a 1 second simulation is less than twice the length of 0.5 second simulations shown in Table 14. The solver seems to make optimizations by converging faster and reducing iteration counts benefiting the overall completion time. Again, it is found that the completion times for all alternatives to be within 5% of each other with a few non-conclusive exceptions.

Table 16 shows the execution times for 0.1 second depthCharge3D experiment. This complex CFD calculation requires 2.5 hours on a single core and 1 hour on 4 cores. The performance of Dockerized HPC is comparable to (and sometimes

Table 15: DepthCharge2D – 1 second simulation results.

	1		2 (simple)		2 (scotch)	
	Iterations	Time (s)	Iterations	Time (s)	Iterations	Time (s)
Physical Machine	5854	1021.21	5759	694.3	5642	571.5
Physical+Docker	5854	1026.24	5759	601.61	5642	594.8
Virtual Machine	5854	1059.3	5759	609.71	5642	604.65

slightly better than) both Physical and Virtual HPC setups. Therefore, the conclusion reached is that there is no significant performance difference among Physical, Virtual and Dockerized HPC choices and the real improvements can be achieved within multi-core hardware and HPC solvers and software.

Table 16: DepthCharge 3D - 0.1 second simulation results.

	1	2 (Simple)	2 (Scotch)	4 (Simple)	4 (Scotch)
	Time (min:sec)	Time (min:sec)	Time (min:sec)	Time (min:sec)	Time (min:sec)
Physical Machine	146:28	79:03	82:15	58:02	61:53
Physical + Docker	147:40	83:20	92:22	56:39	62:07
Virtual Machine	151:45	80:35	86:33	52:18	60:02

4.4.0.1 Isolation Tests and Results

In Table 17, the isolation test results can be seen. Two DepthCharge2D tasks using 4-core each over an 8-core server. In this experiment, two depthCharge2D loads (Load1 & Load2) with scotch decomposition on the same 8-core machine are executed by using 4-cores for each load, to understand whether there are interactions among loads and whether Docker can provide the desired isolation properties. Table 17 shows the execution times with real, user, system breakdowns. The most important learning from these results is that, loads can slow each other by 10-11% when executed together over Docker, but this slowdown can mainly be attributed to the switching costs in the kernel as seen in the 28-30% increase in the "system" time. This observation triggers other OS level studies for HPC, therefore the future work will include 1-changing priorities of jobs to observe the effectiveness of OS in differentiating them and 2-changing the default time-sharing, fair preemptive scheduling of Ubuntu Linux into Batch, FIFO, Round-Robin (SCHED_BATCH, SCHED_FIFO, SCHED_RR) methods [47]. It must also be noted here that when Load1 and Load2 where dispatched onto a 4-core machine with the option to use 4-cores each, the experiments did not complete (i.e. experiments are terminated after 4 hours). Therefore, the container world should also benefit from automated job schedulers, or the burden of careful workload planning would be on the programmers' or cluster-owners' side.

Table 17: Isolation test results. Two DepthCharge2D tasks using 4-core each over an 8-core server.

	Real (sec)	User-4core (sec)	Sys-4core (sec)
Load1-only	270.0 (ref.)	870.0 (Ref.)	11.5 (Ref.)
Load1 (with L1+L2)	299.3 (+%10.8)	924.6 (+%6.3)	15.0 (+%30.4)
Load2 (with L1+L2)	300.1 (+%11.1)	907.4 (+%4.3)	14.8 (+%28.7)

Finally, Docker application container performance is compared with Physical machines and VM, which are the commonly use HPCaaS infrastructure alternatives. As the CPU-intensive benchmark, OpenFOAM computational fluid dynamics (CFD) software is used and as for the IO-intensive benchmarks, Hadoop teragen and terasort with different job sizes (1-40GB) and HDFS block sizes (32-64-128 MB) (Figures 32 and 33) are used.

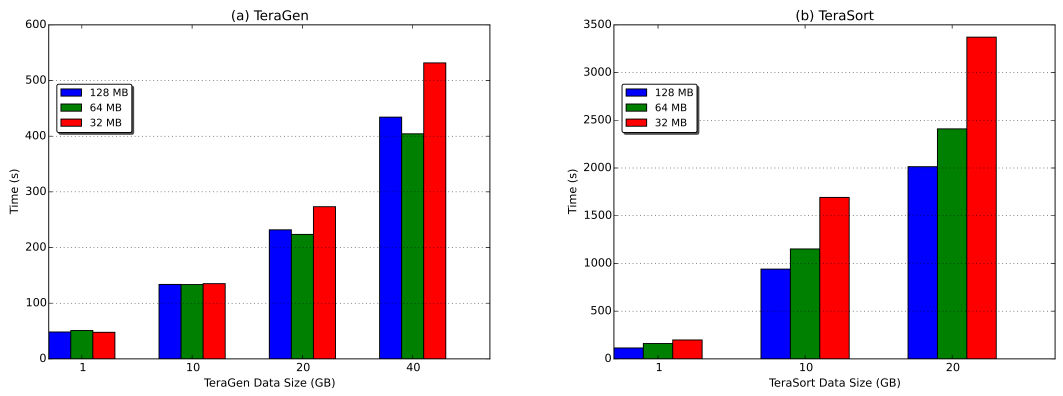


Figure 32: Physical Servers: Hadoop TeraGen and TeraSort benchmarks with different data (1-40GB) and block sizes (32MB-128MB).

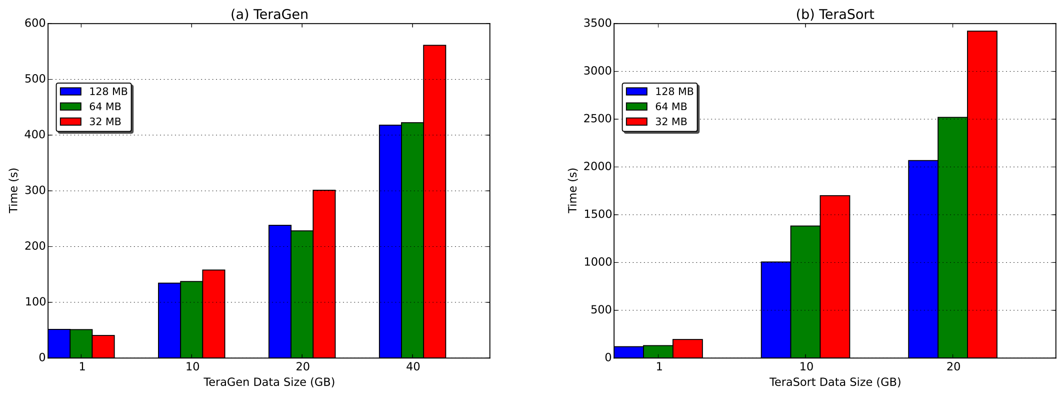


Figure 33: Docker Servers: Hadoop TeraGen and TeraSort benchmarks with different data (1-40GB) and block sizes (32MB-128MB).

CHAPTER V

CONCLUSION AND FUTURE WORK

In this thesis, the design and implementation of a Finite Element Analysis cloud service that can be used to solve common problems in the High Performance Technical Computing (HPTC) field is described. For these purposes, the CPU and memory requirements of representative structural mechanics workloads are characterized and addressed some of the performance challenges related to concurrent job processing are addressed. An extensive performance benchmarking over heterogeneous multi-core, multi-node computing resources is carried out and it shown that effective job characterization and smart scheduling via automated parameter tuning for effective utilization of CPU and memory resources can result in significant time and throughput improvements. The aim of the study is to simplify use of these FEA tools by a broader community of people including SMBs and academicians without the burden of IT management.

Additionally, to further deepen the understanding of the solver behavior, it is found that iterative solvers with selective Krylov subspace methods such as CG and preconditioning such as BJacobi can deliver better performance than direct solvers for the larger-sized linear mechanical structural analysis jobs. However, in certain niche cases direct solvers –usually smaller sized jobs- can also show better performance. Thus, there is a need for selecting the best solver type and preconditioning combinations in HPC-FEA cloud services for predictable performance and price. The common belief that time and memory are exchangable in HPC (i.e. that you can always buy time by throwing more memory at the problem) is disproven. Because, any serious-sized computational job ($>$ 10 million NZE) will hit the limits of OS memory management. In addition to aforementioned observations, it can claimed that parallelization is only feasible for larger-sized jobs, but none of the jobs in these test cases benefits from ≥ 32 core parallelization.

In this characterization study, it is demonstrated that in addition to the $2x - 3x$ speedups gained from parallelization of FEA tasks, one can gain an additional $2x - 3x$ speedup by carefully setting up of solver types and preconditioner combinations. This study therefore yields notable performance increases towards improving throughput and minimizing job latencies for batch FEA tasks, which were discussed in section 3.1. Additionally, studies on involving condition number, as a parameter into the job execution time prediction, have been evaluated, but gave little return compared to utilization of Non-zero elements.

To summarize, the future work consists of extending the service into areas shown in Figure 2 as "future extensions", handling parallel I/O for bigger FEM files with MPI-IO or MapReduce, more performance analyses using different job mixes, geometries, materials, scheduling algorithms and fully-implementing system features such as privacy-awareness and automated accounting into the FEA service. The knowledge gained through detailed job characterization to design a task-aware, multi-core/multi-node scheduling algorithm that can dynamically select necessary computation parameters, repartition the loads and submit them to the cluster in a resource-aware manner [107] can be very critical in development of online FEA services. Using such domain-specific smart job scheduling algorithms, it becomes possible to improve the utilization, performance and predictability of job executions for HPC-SaaS cloud services.

In this thesis, the performance and isolation characteristics of OpenFOAM CFD software with Docker containers are also tested and results are compared to physical and virtual machines in the Scaleway cloud. It is found that Dockerized HPC can be setup and deployed much faster than pyshical or virtual HPC alternatives and its performance is comparable. While there is still some room for performance improvement of container technologies at the OS level, the potential $5-10\%$ savings would be negligible compared to the $10-15x$ improvements provided by containers in deployment times. If scientific workflows and engineering problems requiring HPC solutions can be quickly setup, tested and then reconfigured and retested, then IT related bottlenecks would be removed allowing engineers to focus on solving their real scientific or sectoral problems. Containers are found to carry this huge potential. This research can be extended with different HPC jobs, on different public clouds, and using different solvers and scenarios to obtain more detailed understanding of Dockerized HPC applications in the Cloud. Additionally, experiments for comparing the performance and isolation characteristics of Native, Dockerized, and Virtualized Cluster models with CPU intensive HPC workloads and I/O-intensive Hadoop workloads have also been executed and reported.

Hopefully, the proposed service will simplify and help the design and wide-scale use of FEA and other scientific and engineering applications including heat transfer, fluid dynamics, acoustics, and electromagnetic modeling in the future.

Bibliography

[1] "Nist definition of cloud computing v.15." `http://csrc.nist.gov/groups/SNS/cloud-computing`.

[2] P. Mell and T. Grance, "The nist definition of cloud computing draft." `http://www.nist.gov/itl/cloud/index.cfm`, January 2011.

[3] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Computer Systems*, vol. 25, no. 6, pp. 599 – 616, 2009.

[4] P. S. Pacheco, *Parallel Programming with MPI*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1996.

[5] "SimScale - CFD, FEA, and Thermal Simulation in the Cloud — CAE." `https://www.simscale.com/`.

[6] G. Shainer, T. Liu, J. Layton, and J. Mora, "Scheduling strategies for hpc as a service (hpcaas)," in *2009 IEEE International Conference on Cluster Computing and Workshops*, pp. 1–6, Aug 2009.

[7] K. Bathe, *Finite Element Procedures*. Prentice Hall, 2006.

[8] S. Ma and L. Tian, "A web service-based multi-disciplinary collaborative simulation platform for complicated product development," *The International Journal of Advanced Manufacturing Technology*, vol. 73, pp. 1033–1047, Jul 2014.

[9] J. Xie, Z. Yang, X. Wang, and X. Lai, "A cloud service platform for the seamless integration of digital design and rapid prototyping manufacturing," *The International Journal of Advanced Manufacturing Technology*, Oct 2018.

[10] D. Wu, X. Liu, S. Hebert, W. Gentzsch, and J. Terpenny, "Democratizing digital design and manufacturing using high performance cloud computing: Performance evaluation and benchmarking," *Journal of Manufacturing Systems*, vol. 43, pp. 316 – 326, 2017. High Performance Computing and Data Analytics for Cyber Manufacturing.

[11] A. Harish, "How to Choose a Solver for FEM Problems: Direct or Iterative." `https://www.simscale.com/blog/2016/08/how-to-choose-solvers-for-fem/`, March 2018.

[12] K. Suresh, "Efficient generation of large-scale pareto-optimal topologies," *Structural and Multidisciplinary Optimization*, vol. 47, pp. 49–61, Jan 2013.

[13] K. Suresh, "Topology optimization on the cloud: a confluence of technologies," in *Proceedings of the ASME 2015 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference IDETC/CIE 2015*, August 2015.

[14] "Amazon elastic computing cloud (ec2) pricing." `https://aws.amazon.com/pricing/`.

[15] S. Ostermann, A. Iosup, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema, "A performance analysis of ec2 cloud computing services for scientific computing," in *Cloud Computing* (D. R. Avresky, M. Diaz, A. Bode, B. Ciciani, and E. Dekel, eds.), (Berlin, Heidelberg), pp. 115–131, Springer Berlin Heidelberg, 2010.

[16] A. B. Nagarajan, F. Mueller, C. Engelmann, and S. L. Scott, "Proactive fault tolerance for hpc with xen virtualization," in *Proceedings of the 21st Annual International Conference on Supercomputing*, ICS '07, (New York, NY, USA), pp. 23–32, ACM, 2007.

[17] P. F. Giffuni, "Applications for the freebsd os in mechanical engineering," tech. rep., 2006.

[18] F. Gherardini, C. Renzi, and F. Leali, "A systematic user-centred framework for engineering product design in small- and medium-sized enterprises (smes)," *The International Journal of Advanced Manufacturing Technology*, vol. 91, pp. 1723–1746, Jul 2017.

[19] N. Muhtaroglu, I. Ari, and E. Koyun, "Towards automatic selection of direct vs. iterative solvers for cloud-based finite element analysis," in *Proceedings of the Fourth International Conference on Parallel, Distributed, Grid and Cloud Computing for Engineering*, Civil-Comp Press, Stirlingshire, Scotland, 2015. CIVIL-COMP: Parallel, Distributed, Grid and Cloud Computing.

[20] "Amazon web services." `http://aws.amazon.com`.

[21] "Top500 supercomputer sites." `http://www.top500.org`.

[22] E. Joseph, S. Conway, and J. Wu, "A new approach to hpc public clouds: The sgi cyclone hpc cloud." `http://www.sgi.com/pdfs/4215.pdf`.

[23] "Sun grid." `http://www.sun.com/service/sungrid/`.

[24] T. Naughton, C. Engelmann, G. Vallee, S. L. Scott, and H. Ong, "System-level virtualization for high performance computing," in *16th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP 2008)(PDP)*, vol. 00, pp. 636–643, 02 2008.

[25] B. Xiaoyong, "High performance computing for finite element in cloud," in *2011 International Conference on Future Computer Sciences and Application*, pp. 51–53, June 2011.

[26] K. M. Zingerman, A. V. Vershinin, and V. A. Levin, "An approach for verification of finite-element analysis in nonlinear elasticity under large strains," *IOP Conference Series: Materials Science and Engineering*, vol. 158, no. 1, p. 012104, 2016.

[27] T.-C. Chiang and L.-C. Fu, "Solving the fms scheduling problem by critical ratio-based heuristics and the genetic algorithm," in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, vol. 3, pp. 3131–3136 Vol.3, April 2004.

[28] M. A. Chik, I. Ahmad, and M. Y. Jamaluddin, "A simulation approach for dispatching techniques comparison in 200mm wafer foundry," in *2004 IEEE International Conference on Semiconductor Electronics*, pp. 5 pp.–, Dec 2004.

[29] Y. Park, K. Casey, and S. Baskiyar, "A novel adaptive support vector machine based task scheduling," in *Proceedings the 9th International Conference on Parallel and Distributed Computing and Networks*, pp. 16–18, February 2010.

[30] A. D. Breslow, L. Porter, A. Tiwari, M. Laurenzano, L. Carrington, D. M. Tullsen, and A. E. Snavely, "The case for colocation of high performance computing workloads," *Concurrency and Computation: Practice and Experience*, vol. 28, no. 2, pp. 232–251.

[31] N. Doulamis, A. Doulamis, A. Litke, A. Panagakis, T. Varvarigou, and E. Varvarigos, "Adjusted fair scheduling and non-linear workload prediction for qos guarantees in grid computing," *Computer Communications*, vol. 30, no. 3, pp. 499 – 515, 2007. Special Issue: Emerging Middleware for Next Generation Networks.

[32] T. Belytschko and K. Mish, "Computability in non-linear solid mechanics," *International Journal for Numerical Methods in Engineering*, vol. 52, no. 1-2, pp. 3–21, 2001.

[33] A. K. Amoura, E. Bampis, C. Kenyon, and Y. Manoussakis, "Scheduling independent multiprocessor tasks," in *Algorithms — ESA '97* (R. Burkard and G. Woeginger, eds.), (Berlin, Heidelberg), pp. 1–12, Springer Berlin Heidelberg, 1997.

[34] J. Dümmler, T. Rauber, and G. Rünger, "Scalable computing with parallel tasks," in *Proceedings of the 2Nd Workshop on Many-Task Computing on Grids and Supercomputers*, MTAGS '09, (New York, NY, USA), pp. 9:1–9:10, ACM, 2009.

[35] I. Ari and N. Muhtaroglu, "Design and implementation of a cloud computing service for finite element analysis," *Advances in Engineering Software*, vol. 60-61, pp. 122 – 135, 2013. CIVIL-COMP: Parallel, Distributed, Grid and Cloud Computing.

[36] J. C. Briggs, "Developing an architecture framework for cloud-based, multi-user, finite element pre-processing," 2013.

[37] "OpenFOAM tutorial." `http://www.cfd.at/downloads/2014_OFoam_Tut_Complete.pdf`, February 2015.

[38] G. Dhondt, "Calculix crunchix user's manual v2.0." `http://www.calculix.de`.

[39] E. de France, "Finite element *code_aster*, analysis of structures and thermomechanics for studies and research, year = 1989–2017." Open source on www.code-aster.org.

[40] G. Bui and G. Meschke, "Study on performance of parallel solvers for coupled simulations of partially saturated soils in tunnel engineering application," in *Proceedings of the Fourth International Conference on Parallel, Distributed, Grid and Cloud Computing for Engineering*, Civil-Comp Press, Stirlingshire, Scotland, 2015. CIVIL-COMP: Parallel, Distributed, Grid and Cloud Computing.

[41] C. Pommerell and W. Fichtner, "Memory aspects and performance of iterative solvers," *SIAM Journal on Scientific Computing*, vol. 15, no. 2, pp. 460–473, 1994.

[42] F. Sourbier, A. Haidar, L. Giraud, H. Ben-Hadj-Ali, S. Operto, and J. Virieux, "Three-dimensional parallel frequency-domain visco-acoustic wave modelling based on a hybrid direct/iterative solver," *Geophysical Prospecting*, vol. 59, pp. 834–856, August 2011.

[43] M. P. Bendsoe and O. Sigmund, *Topology Optimization: Theory, Methods and Applications*. Springer, Feb. 2004.

[44] P. Menage, "Cgroups." `https://www.kernel.org/doc/Documentation/cgroup-v1/cgroups.txt`.

[45] R. L. Henderson, "Job scheduling under the portable batch system," in *Job Scheduling Strategies for Parallel Processing* (D. G. Feitelson and L. Rudolph, eds.), (Berlin, Heidelberg), pp. 279–294, Springer Berlin Heidelberg, 1995.

[46] A. B. Yoo, M. A. Jette, and M. Grondona, "Slurm: Simple linux utility for resource management," in *Job Scheduling Strategies for Parallel Processing* (D. Feitelson, L. Rudolph, and U. Schwiegelshohn, eds.), (Berlin, Heidelberg), pp. 44–60, Springer Berlin Heidelberg, 2003.

[47] B. Ruan, H. Huang, S. Wu, and H. Jin, “A performance study of containers in cloud environment,” in *Advances in Services Computing* (G. Wang, Y. Han, and G. Martínez Pérez, eds.), (Cham), pp. 343–356, Springer International Publishing, 2016.

[48] M. G. Xavier, M. V. Neves, F. D. Rossi, T. C. Ferreto, T. Lange, and C. A. F. D. Rose, “Performance evaluation of container-based virtualization for high performance computing environments,” in *2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, pp. 233–240, February 2013.

[49] D. Merkel, “Docker: Lightweight linux containers for consistent development and deployment,” *Linux J.*, vol. 2014, Mar. 2014.

[50] L. Gerhardt, W. Bhimji, S. Canon, M. Fasel, D. Jacobsen, M. Mustafa, J. Porter, and V. Tsulaia, “Shifter: Containers for HPC,” *Journal of Physics: Conference Series*, vol. 898, no. 8, p. 082021, 2017.

[51] W. Gerlach, W. Tang, K. Keegan, T. Harrison, A. Wilke, J. Bischof, M. D’Souza, S. Devoid, D. Murphy-Olson, N. Desai, and F. Meyer, “Skyport: Container-based execution environment management for multi-cloud scientific workflows,” in *Proceedings of the 5th International Workshop on Data-Intensive Computing in the Clouds*, DataCloud ’14, (Piscataway, NJ, USA), pp. 25–32, IEEE Press, 2014.

[52] Z. Kozhirbayev and R. O. Sinnott, “A performance comparison of container-based technologies for the cloud,” *Future Generation Computer Systems*, vol. 68, pp. 175 – 182, 2017.

[53] A. Y.-Z. Ou and J.-C. Chen, “Head-to-Head: Which is the Better Cloud Platform for Early Stage Start-up? Docker versus OpenStack.,” tech. rep., Department of Computer Science, University of Illinois at Urbana-Champaign, May 2015.

[54] Y. Babu, “Docker Container Cluster Deployment Across Different Networks,” Master’s thesis, National College of Ireland, Dublin, Ireland, 2016.

[55] “Syscalls for scheduling.” `http://manpages.ubuntu.com/manpages/zesty/en/man2/sched_setscheduler.2.html`.

[56] J.-l. Lafragette and B. Orlandi, “A network, a cloud-based server, and a method of registering for a service.” `http://www.freepatentsonline.com/y2018/0167354.html`, June 2018.

[57] C. Kaewkasi, “Docker for serverless applications: Containerize and orchestrate functions using openfaas, openwhisk, and fn,” 2018.

[58] C. Bargmann, “Serverless & faas,” 2018.

[59] T. Bajarin, "Is cloud computing the next big thing?." http://www.pcmag.com/article2/0,2817,2277819,00.asp.

[60] V. Sekar and P. Maniatis, "Verifiable resource accounting for cloud computing services," in *Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop*, CCSW '11, (New York, NY, USA), pp. 21–26, ACM, 2011.

[61] J. Li, I. Ari, J. Jain, A. H. Karp, and M. Dekhil, "Mobile in-store personalized services," in *2009 IEEE International Conference on Web Services*, pp. 727–734, July 2009.

[62] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud computing and grid computing 360-degree compared," in *2008 Grid Computing Environments Workshop*, pp. 1–10, November 2008.

[63] "Open cloud computing interface standard by open grid forum occi working group." http://occi-wg.org/.

[64] "Dassault systemes plm solutions and catia design." http://www.3ds.com/.

[65] D. Bremberg and G. Dhondt, "Automatic crack-insertion for arbitrary crack growth," *Engineering Fracture Mechanics*, vol. 75, pp. 404 – 416, February - March 2008. International Conference of Crack Paths.

[66] J. Schöberl, "Netgen an advancing front 2d/3d-mesh generator based on abstract rules," *Computing and Visualization in Science*, vol. 1, pp. 41–52, Jul 1997.

[67] "Web app framework." http://en.wikipedia.org/wiki/List_of_web_application_frameworks.

[68] K. Wittig, "Calculix user's manual - calculix graphix v.2.0." http://www.bconverged.com/calculix/doc/cgx/.

[69] D. Flanagan, *JavaScript - The Definitive Guide: Activate Your Web Pages: Covers ECMAScript 5 and HTML5 (6. ed.).* O'Reilly, 2011.

[70] "Webgl by khronos group." http://www.khronos.org/webgl/.

[71] C. Ashcraft, D. Pierce, D. Wah, and J. Wu, "The reference manual for spooles, release 2.2: An oo software library for solving sparse linear systems of equations." http://www.netlib.org/linalg/spooles/.

[72] O. Schenk, K. Gärtner, W. Fichtner, and A. Stricker, "Pardiso: a high-performance serial and parallel sparse linear solver in semiconductor device simulation," *Future Generation Computer Systems*, vol. 18, no. 1, pp. 69 – 78, 2001. I. High Performance Numerical Methods and Applications. II. Performance Data Mining: Automated Diagnosis, Adaption, and Optimization.

[73] S. Hsieh, Y. Yang, and P. Hsu, "Integration of general sparse matrix and parallel computing technologies for large–scale structural analysis," *Computer-Aided Civil and Infrastructure Engineering*, vol. 17, pp. 423–438, December 2002.

[74] G. Karypis and V. Kumar, "Metis – unstructured graph partitioning and sparse matrix ordering system, version 2.0," tech. rep., 1995.

[75] "Matlab sparse matrix operations." `http://www.mathworks.com/help/techdoc/math`.

[76] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, pp. 107–113, January 2008.

[77] "Amdahl's law speedup chart on wikipedia." `http://en.wikipedia.org/wiki/Andahl's_law`.

[78] L. V. Tsap, D. B. Goldgof, S. Sarkar, and W.-C. Huang, "Efficient nonlinear finite element modeling of nonrigid objects via optimization of mesh models," *Computer Vision and Image Understanding*, vol. 69, no. 3, pp. 330 – 350, 1998.

[79] J. Seward, N. Nethercote, and J. Weidendorfer, *Valgrind 3.3 - Advanced Debugging and Profiling for GNU/Linux Applications.* Network Theory Ltd., 2008.

[80] J. Howard, S. Dighe, Y. Hoskote, S. Vangal, D. Finan, G. Ruhl, D. Jenkins, H. Wilson, N. Borkar, G. Schrom, F. Pailet, S. Jain, T. Jacob, S. Yada, S. Marella, P. Salihundam, V. Erraguntla, M. Konow, M. Riepen, G. Droege, J. Lindemann, M. Gries, T. Apel, K. Henriss, T. Lund-Larsen, S. Steibl, S. Borkar, V. De, R. V. D. Wijngaart, and T. Mattson, "A 48-core ia-32 message-passing processor with dvfs in 45nm cmos," in *2010 IEEE International Solid-State Circuits Conference - (ISSCC)*, pp. 108–109, February 2010.

[81] A. B. Mohammed, J. Altmann, and J. Hwang, *Cloud Computing Value Chains: Understanding Businesses and Value Creation in the Cloud*, pp. 187–208. Basel: Birkhäuser Basel, 2010.

[82] F. A. Alali and C.-L. Yeh, "Cloud computing: Overview and risk analysis," *Journal of Information Systems*, vol. 26, no. 2, pp. 13–33, 2012.

[83] B. Martens, M. Walterbusch, and F. Teuteberg, "Costing of cloud computing services: A total cost of ownership approach," in *2012 45th Hawaii International Conference on System Sciences*, pp. 1563–1572, Jan 2012.

[84] B. Smith, *PETSc (Portable, Extensible Toolkit for Scientific Computation)*, pp. 1530–1539. Boston, MA: Springer US, 2011.

[85] P. Amestoy, I. Duff, and J.-Y. L'Excellent, "Mumps multifrontal massively parallel solver version 2.0," 1998.

[86] V. Simoncini and D. B. Szyld, "Recent computational developments in krylov subspace methods for linear systems," *Numerical Linear Algebra with Applications*, vol. 14, no. 1, pp. 1–59.

[87] Y. Saad, *Iterative Methods for Sparse Linear Systems.* Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2nd ed., 2003.

[88] M. H. Gutknecht, "A brief introduction to krylov space methods for solving linear systems," in *Frontiers of Computational Science* (Y. Kaneda, H. Kawamura, and M. Sasai, eds.), (Berlin, Heidelberg), pp. 53–62, Springer Berlin Heidelberg, 2007.

[89] J. Chen, Y. Lee, and S. Shieh, "Revisit of two classical elasticity problems by using the trefftz method," *Engineering Analysis with Boundary Elements*, vol. 33, no. 6, pp. 890 – 895, 2009.

[90] I. S. Duff and J. K. Reid, "The multifrontal solution of indefinite sparse symmetric linear," *ACM Trans. Math. Softw.*, vol. 9, pp. 302–325, Sept. 1983.

[91] M. Benzi and M. Tûma, "A comparative study of sparse approximate inverse preconditioners," *Applied Numerical Mathematics*, vol. 30, no. 2, pp. 305 – 340, 1999.

[92] E. Agullo, L. Giraud, and M. Zounon, "On the resilience of parallel sparse hybrid solvers," in *2015 IEEE 22nd International Conference on High Performance Computing (HiPC)*, pp. 75–84, December 2015.

[93] W. Liu, A. Li, J. D. Hogg, I. S. Duff, and B. Vinter, "Fast synchronization-free algorithms for parallel sparse triangular solves with multiple right-hand sides," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 21, p. e4244. e4244 cpe.4244.

[94] E. Chow, H. Anzt, J. Scott, and J. Dongarra, "Using jacobi iterations and blocking for solving sparse triangular systems in incomplete factorization preconditioning," *Journal of Parallel and Distributed Computing*, vol. 119, pp. 219 – 230, 2018.

[95] S. Timoshenko and J. Goodier, *Theory of Elasticity.* New York: McGraw-Hill, 1970.

[96] T. Blacker, W. Bohnhoff, and T. Edwards, "Cubit mesh generation environment. volume 1: Users manual," 5 1994.

[97] M. Gorman, *Understanding the Linux Virtual Memory Manager.* Upper Saddle River, NJ, USA: Prentice Hall PTR, 2004.

[98] A. Cline, C. Moler, G. Stewart, and J. Wilkinson, "An estimate for the condition number of a matrix," *SIAM Journal on Numerical Analysis*, vol. 16, no. 2, pp. 368–375, 1979.

[99] C. A. Waldspurger, “Memory resource management in vmware esx server,” *SIGOPS Oper. Syst. Rev.*, vol. 36, pp. 181–194, Dec. 2002.

[100] “Microsoft hyper-v server virtualization.” `https://www.microsoft.com/en-us/cloud-platform/server-virtualization`.

[101] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, “An updated performance comparison of virtual machines and linux containers,” in *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 171–172, March 2015.

[102] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, “Xen and the art of virtualization,” *SIGOPS Oper. Syst. Rev.*, vol. 37, pp. 164–177, October 2003.

[103] D. Liu and L. Zhao, “The research and implementation of cloud computing platform based on docker,” in *2014 11th International Computer Conference on Wavelet Actiev Media Technology and Information Processing(ICCWAMTIP)*, pp. 475–478, December 2014.

[104] F. Pellegrini and J. Roman, “Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs,” in *High-Performance Computing and Networking* (H. Liddell, A. Colbrook, B. Hertzberger, and P. Sloot, eds.), (Berlin, Heidelberg), pp. 493–498, Springer Berlin Heidelberg, 1996.

[105] S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith, *Efficient Management of Parallelism in Object-Oriented Numerical Software Libraries*, pp. 163–202. Boston, MA: Birkhäuser Boston, 1997.

[106] N. Muhtaroglu, B. Kolcu, and I. Ari, “Testing performance of application containers in the cloud with hpc loads,” in *Proceedings of the Fifth International Conference on Parallel, Distributed, Grid and Cloud Computing for Engineering*, Civil-Comp Press, Stirlingshire, Scotland, 2017. CIVIL-COMP: Parallel, Distributed, Grid and Cloud Computing.

[107] I. Ari and U. Kocak, “Hybrid job scheduling for improved cluster utilization,” in *Euro-Par 2013: Parallel Processing Workshops* (D. an Mey, M. Alexander, P. Bientinesi, M. Cannataro, C. Clauss, A. Costan, G. Kecskemeti, C. Morin, L. Ricci, J. Sahuquillo, M. Schulz, V. Scarano, S. L. Scott, and J. Weidendorfer, eds.), (Berlin, Heidelberg), pp. 395–405, Springer Berlin Heidelberg, 2014.

VITA

Nitel Muhtaroglu is a Ph.D. student at Ozyegin University Computer Science Department studying under supervision of Dr. Ari. He also holds the title of Sr. Staff Software Engineer at GE Aviation. He received his M.Sc. degree in Computational Engineering from Ruhr-University Bochum in 2005 and B.Sc. degree in Mechanical Engineering from Istanbul University in 2001. Previously, he carried out research at Faculty of Mechanics and Mathematics of Moscow State University. His research interests mainly focus on Computational Mechanics, Engineering Informatics, High Performance Computing and Applied Mathematics.