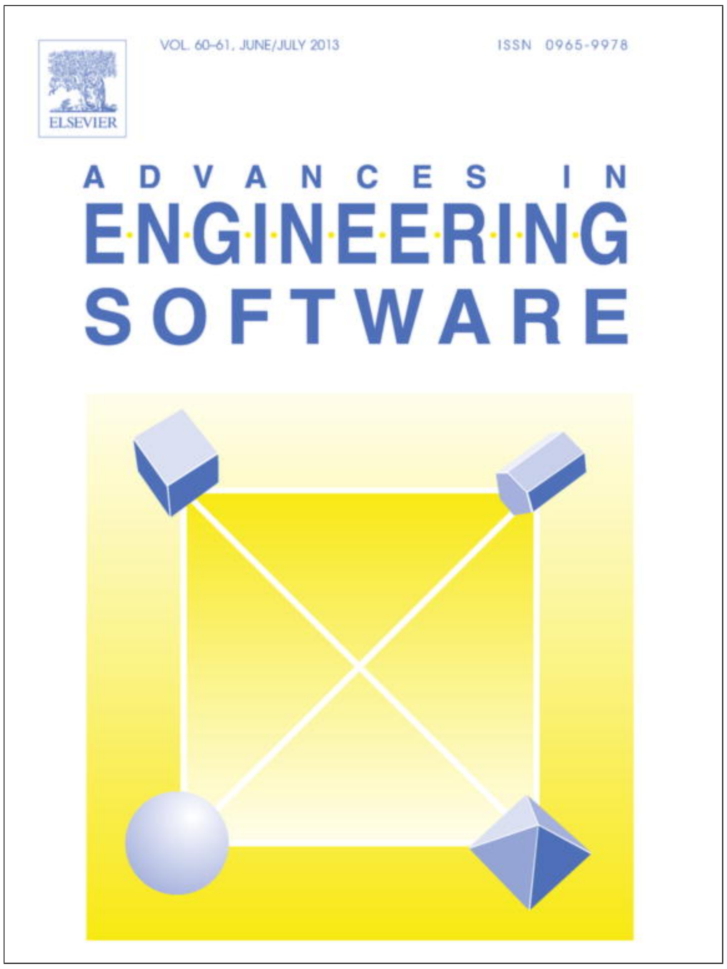

VOL. 60–61, JUNE/JULY 2013
ISSN 0965-9978
ELSEVIER
ADVANCES IN ENGINEERING SOFTWARE

Advances in Engineering Software 60-61 (2013) 122–135

Contents lists available at SciVerse ScienceDirect

Advances in Engineering Software

journal homepage: www.elsevier.com/locate/advengsoft

# Design and implementation of a cloud computing service for finite element analysis

Ismail Ari [*], Nitel Muhtaroglu

*Computer Science Department, Ozyegin University, Istanbul, Turkey*

ARTICLE INFO

*Article history:*
Available online 8 November 2012

*Keywords:*
Cloud computing
Finite element analysis
Structural mechanics
Task scheduling
Multi-core
SPOOLES

ABSTRACT

This paper presents 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). The focus is specifically on performance characterization of linear and nonlinear mechanical structural analysis workloads over multi-core and multi-node computing resources. We first analyze and observe that accurate job characterization, tuning of multi-threading parameters and effective multi-core/node scheduling are critical for service performance. We design a "smart" scheduler that can dynamically select some of the required parameters, partition the load and schedule it in a resource-aware manner. We can achieve up to 7.53× performance improvement over an aggressive scheduler using mixed FEA loads. We also discuss critical issues related to the data privacy, security, accounting, and portability of the cloud service.

## 1. Introduction

According to the U.S. National Institute of Standards and Technology (NIST) [1]: "Cloud Computing is a model for enabling ubiquitous, 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 Fig. 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.

Our experiences with the engineering and scientific communities revealed us 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. 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. 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 FEA.

FEA is a generally-applicable numerical method to approximately solve partial differential equations and requires HPC setups. Fig. 2 shows some of the application areas of FEA including mechanical structural analysis, heat transfer, fluid dynamics, acoustics, and electromagnetic modeling. Several other related numerical methods have been developed in the past (FEM, FDM, FVM, BEM shown in Fig. 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 perform numerical methods are available in the market as desktop or mainframe applications. Some of

* Corresponding author.
*E-mail address:* ismail.ari@ozyegin.edu.tr (I. Ari).

0965-9978/$ - see front matter © 2012 Civil-Comp Ltd and Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.advengsoft.2012.10.003

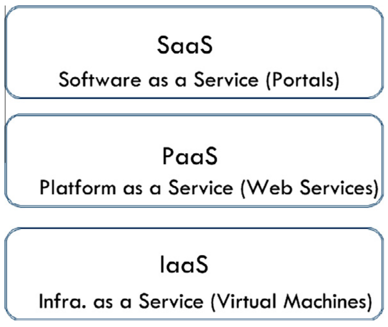

SaaS
Software as a Service (Portals)
PaaS
Platform as a Service (Web Services)
IaaS
Infra. as a Service (Virtual Machines)

**Fig. 1.** Different service models for cloud computing and the logical layering among them.

the well-known proprietary packages include Nastran, Ansys, Abaqus, and open-sources include CalculiX, Code Aster, and various others. However, the installation and large-scale maintenance of these FEA 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 small-medium businesses (SMB) as well as technical individuals, we decided to provide FEA as a cloud computing service. All that the users will need is a personal computing device with a browser and an Internet connection to enable them to access our HPC cloud service for FEA.

Other components shown Fig. 2 are described in more detail later in Section 2.

To sustain high-performance in our FEA service, we first need to accurately characterize our candidate workloads. As FEA is a broad area of research, in this paper we only focus on mechanical structural analysis, which is used ubiquitously in automotive, aviation, home appliance production, construction and defense industries as well as academia. The lessons learned will be generally applicable to other FEA and HPC subject areas, since the underlying mathematical principles are similar. In this paper, we test our structural mechanics benchmarks using open source software tools over local physical servers. In the future, we plan to extend our work into other application areas, methods, solvers and hybrid processing and deployment models [2] shown in Fig. 2. Our current contributions can be listed as follows:

- Design and implementation of a new online FEA cloud service different from existing offerings. Our 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 (beams, rotors, etc.) and their mixes over shared memory (multi-core) and distributed memory (multi-node) resources.
- A comprehensive evaluation of alternative task execution and scheduling strategies and showing performance improvements using smart scheduling.
- Discussions about the 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.

The rest of the paper is organized as follows. Section 2 describes the design of our FEA service architecture. Section 3 characterizes the benchmark workloads used and discusses the differences between linear and nonlinear analysis types. Section 4 describes the experimental setup for performance analysis and gives detailed results. Section 5 presents other important issues for the success of cloud computing services. Section 6 summarizes related and future work and Section 7 concludes the paper.

## 2. FEA service architectural design

A wide variety of sectors deal with mechanical structural analysis problems. In these sectors a rigorous structural evaluation of

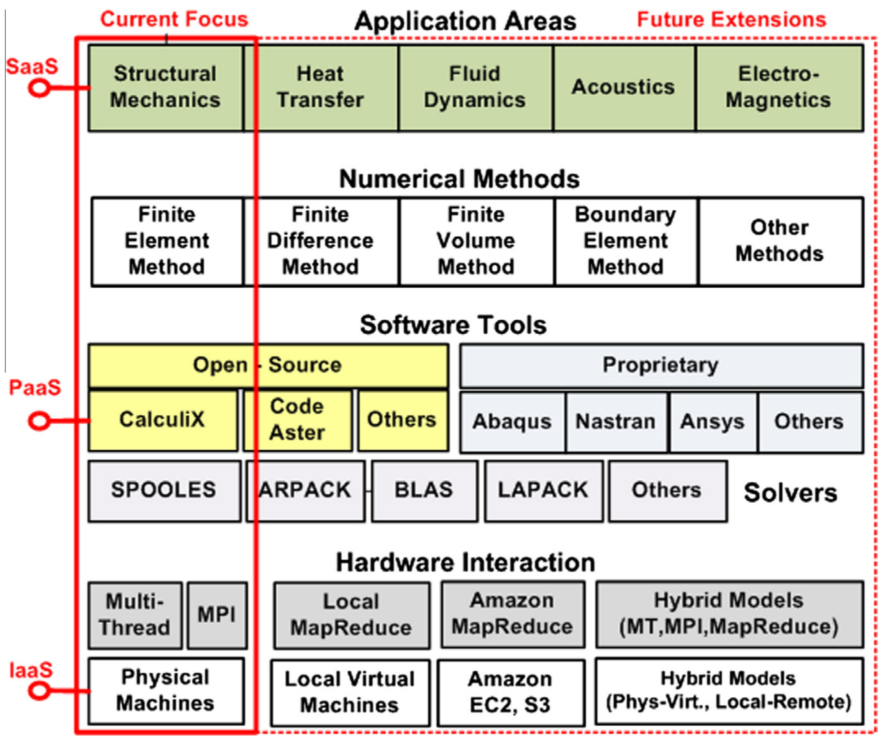


**Fig. 2.** Logical layers and components of a modern FEA cloud service.

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 [3] and increases the reliability of the produced parts reducing the possibility of recalls and critical failures [4]. 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.

Fig. 3 shows the architectural design and some of the implementation details of our FEA service. It consists of the web portal, pre-processor, job scheduler, solvers, and post-processor components in their respective order of execution. We now give a brief discussion about the Computer Aided Design (CAD) process and relate its steps to the components of our service:

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 (STEP) [4]. We currently import the "STL" format designed for rapid 3D *ST*ereo-*L*ithographical prototyping to provide us surface geometry information. To obtain a realistic Finite Element Model (FEM)[1] from the CAD file, a pre-processor tool (such as NetGen [5]) 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 (e.g. INP) that can be processed by the FEA solvers.

### 2.1. Web portal

Web portals[2] such as Liferay, Drupal, and Joomla [6] 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 via the portal. The files uploaded can be raw CAD files or pre-processed mesh (e.g. INP) 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. We are currently using the Java-based Liferay portal because of its ease of integration with other web technologies and the other components of our FEA service.

### 2.2. Pre-processor and solver

We currently use CalculiX [7] as the solver for our online FEA service, because of its open-source availability, wide-adoption in the community and extensive support for solving different engineering problems (see Appendix A for details). CalculiX package has a separate pre-processing tool called CGX (CalculiX GraphiX) [8] that can be used to read and transform the contents of various portable CAD files into a FEM. In our service design, we will allow the pre-and-post processing steps to be done either (1) offline with desktop tools such as NetGen, FEMAP and CGX, or (2) offline inside the web browser's Javascript 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

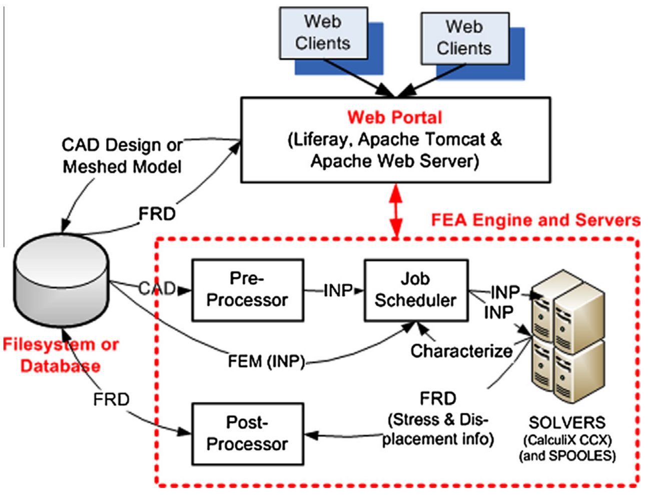


**Fig. 3.** Our FEA cloud service architecture. A preprocessor tool will transform an uploaded CAD file by adding mesh information, material properties, loading type and other necessary computational information to it and turn it into an INP file that is ready for FEA by CalculiX. FRD is a specially formatted file containing CalculiX results.

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 [9]." Fig. 4 shows a screenshot of the 3D viewing of a meshed structure inside the portal of our web site. Canvas element together with the WebGL API can enable us to interact with (select, rotate, zoom, etc.) the 3D objects especially in the pre- and post-processing phases of the design.

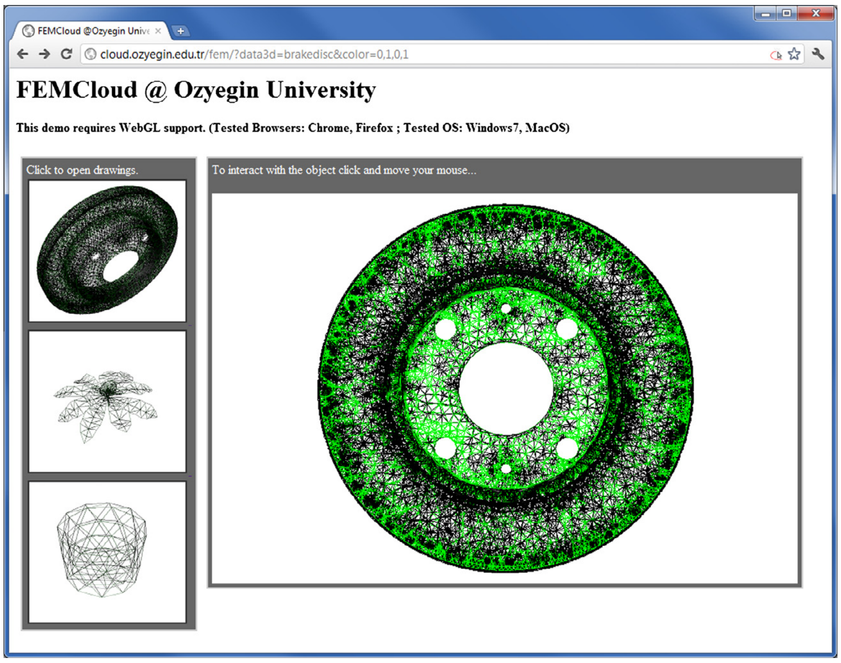
The FEM is consequently converted into a large sparse matrix by CCX (CalculiX CrunchiX) [7] representing the system of linear equations and solved by the underlying solvers such as SPOOLES (Sparse Object-Oriented Linear Equation Solver) [10]. Results obtained help us to accurately estimate the physical displacements, stresses and strains on the structure under applied forces. Several other open-source or proprietary linear equation solvers (PARDISO, TAUCS) can also be used together with CalculiX [7]. We used SPOOLES direct solver in this paper; therefore we skip details for other solvers for brevity. There are also tools for sub-structuring objects before executing the FEA such as METIS and its parallel version PARMETIS. METIS is used for partitioning graphs and finite element meshes, and producing fill reducing orderings for sparse matrices. We currently do not include a sub-structuring (aka domain decomposition) tool in our design for two reasons: (1) research shows that [11] 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 we will see in Section 5 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 $[\boldsymbol{K}] \cdot \{\boldsymbol{u}\} = \{\boldsymbol{f}\}$, is essential in both linear and nonlinear, static and dynamic FEA [11]. In the context of structural mechanics, $\{\boldsymbol{u}\}$ is related to the displacements of each finite element. SPOOLES has four major calculation steps:

- *Communicate*: Read $\boldsymbol{K}$ and $\boldsymbol{f}$ matrices.
- *Reorder*: $(P\boldsymbol{K}P^T) \cdot (P \cdot \boldsymbol{u}) = Pf$.

[1] We use the abbreviation FEM to refer to both the finite element "model" and "method" in this paper. Please refer to the context for the correct meaning.

[2] Web portals are also known as Content Management Systems and they get support from Web Application Frameworks for common activities in web development.


FEMCloud @ Ozyegin University

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

FEMCloud @Ozyegin Univ
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...

**Fig. 4.** A screenshot from the online pre-processing step for our 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.

- *Factorize*: Apply lower–upper (LU) factorization.
- *Solve*: Forward and Backward substitutions.

SPOOLES can be executed in a serial (single-threaded), multi-threaded (pthreads) or multi-node (MPI) fashion [10], 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.

### 2.3. Post-processor

Post-processing can also be done online or offline similar to pre-processing. 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 Fig. 5.

### 2.4. 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 multi-variate 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. We present our 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) and discuss different scheduling techniques in the following sections.

## 3. Workload characterization

In this paper, we used the models shown in Fig. 5 and several others to guide our performance tests and the FEA service design. We chose these models because of the differences and some controlled similarities in their processing complexities. The first is an 8 m × 1 m × 1 m concrete cantilever beam under a 9 MN bending force applied at its free end (i.e. a civil engineering case). The second is a steel 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. 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 <6 MB) 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. Our future work includes handling parallel I/O for bigger FEM files with MPI-IO or using distributed task processing systems such as MapReduce [12] for this purpose. We use MapReduce in our other cloud projects to process 100s of GB of enterprise log files and therefore think that we can apply it to parallelize FEA I/O loads as well.

Other processing-related FEM parameters include number of elements (cubes, tetrahedrons, etc.), integration points for each element, and the number of nodes (or total points). We found that most of these parameters 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. Sparse direct linear equations solvers such as SPOOLES can take advantage of this fact to obtain a compact, memory-efficient representation of the FEM. We observe in our results (Section 4) that both the computational complexity and the memory requirements for the mechanical structural analyses done in this paper are positively correlated with the number of non-zero elements (NZE) in the matrix (e.g. see Fig. 6), which was also indicated by prior

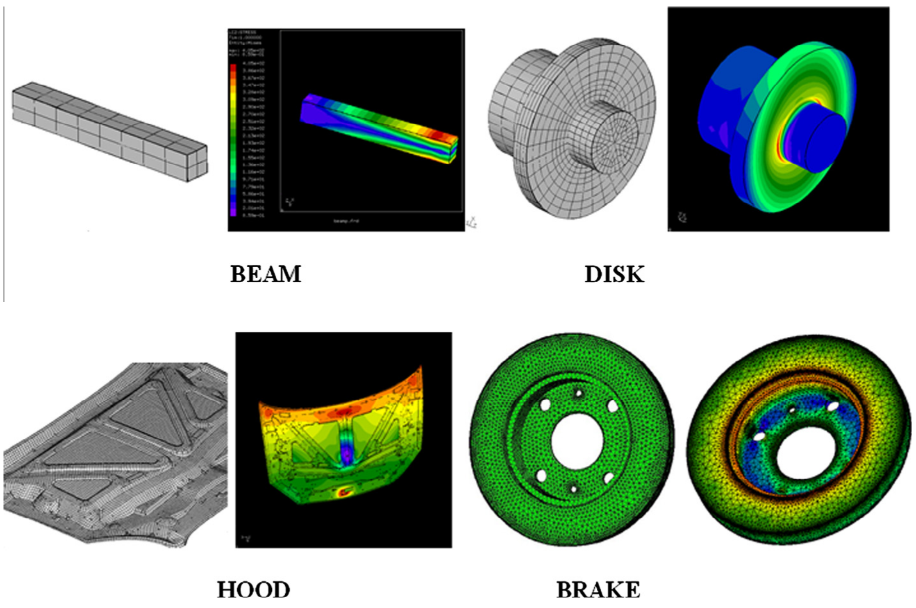


**Fig. 5.** Screenshots from FEA of Beam, Disk, Hood, Brake objects under dynamic forces. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

related work [13]. However, note that there can be counterexamples to this rule for FEA of objects with drastically different geometries, materials and analysis types. Section 4.6 discusses one such scenario for the effect of geometry on performance.

The NZE count and sparsity (i.e. % NZE/Total Elements) of our 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%). We vary the NZE of the Brake component from 8 to 55 million via controlled fine-granularity meshing and measure its effects on memory and CPU time in Section 4 (see Table 7).

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 (these values are also very common in FEA) will have a small incremental performance impact [14]. We processed these FEA jobs 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. We timed the code and measured the parallel portion to span 60–70% of the overall execution time. The results confirmed the validity of Amdahl's Law for these CPU-intensive HPC loads (i.e. increasing the core count is beneficial, but has diminishing returns) and therefore we skip details for brevity. However, we also encountered cases where some large NZE jobs triggered swapping (with kswapd in Linux) due to lack of memory, 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 4 will present detailed experimental results.

### 3.1. 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 and the applied forces also affect the geometry, or the material behavior is nonlinear [27]. We will only focus on the geometric nonlinearity in this paper (see Appendix A for other types of analyses supported by CalculiX).

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 found. Otherwise, the iteration will continue until either convergence or a maximum iteration count is reached. If the maximum iteration is reached, the solution has not converged.

## 4. Experiments

In this section, we first describe our experimental setup for cloud performance analysis (Section 4.1) and present a systematic analysis of the described FEM loads. Since our FEA service will be a multi-tenant, concurrent job processing system we 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, we test the limits of concurrent processing by scheduling $J$ identical jobs each using a single thread (Section 4.2). Third, we increase the number of threads $T$ for each job to better utilize the C cores in each node (Section 4.3). Then, we run jobs that cannot be sustained on a single node over multiple nodes using MPI to benefit from additional distributed computing resources (CPU and memory) (Section 4.4). Next, we mix different jobs and compare different scheduling techniques and show advantages of smart scheduling that takes advantages of adaptive parameter tuning (Section 4.5). Finally, we investigate the effects of structure geometry on the analysis performance (Section 4.6).

### 4.1. Experimental setup

We use two types of systems for our experiments. The first is a PC cluster consisting of 8 × HP DC5850 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

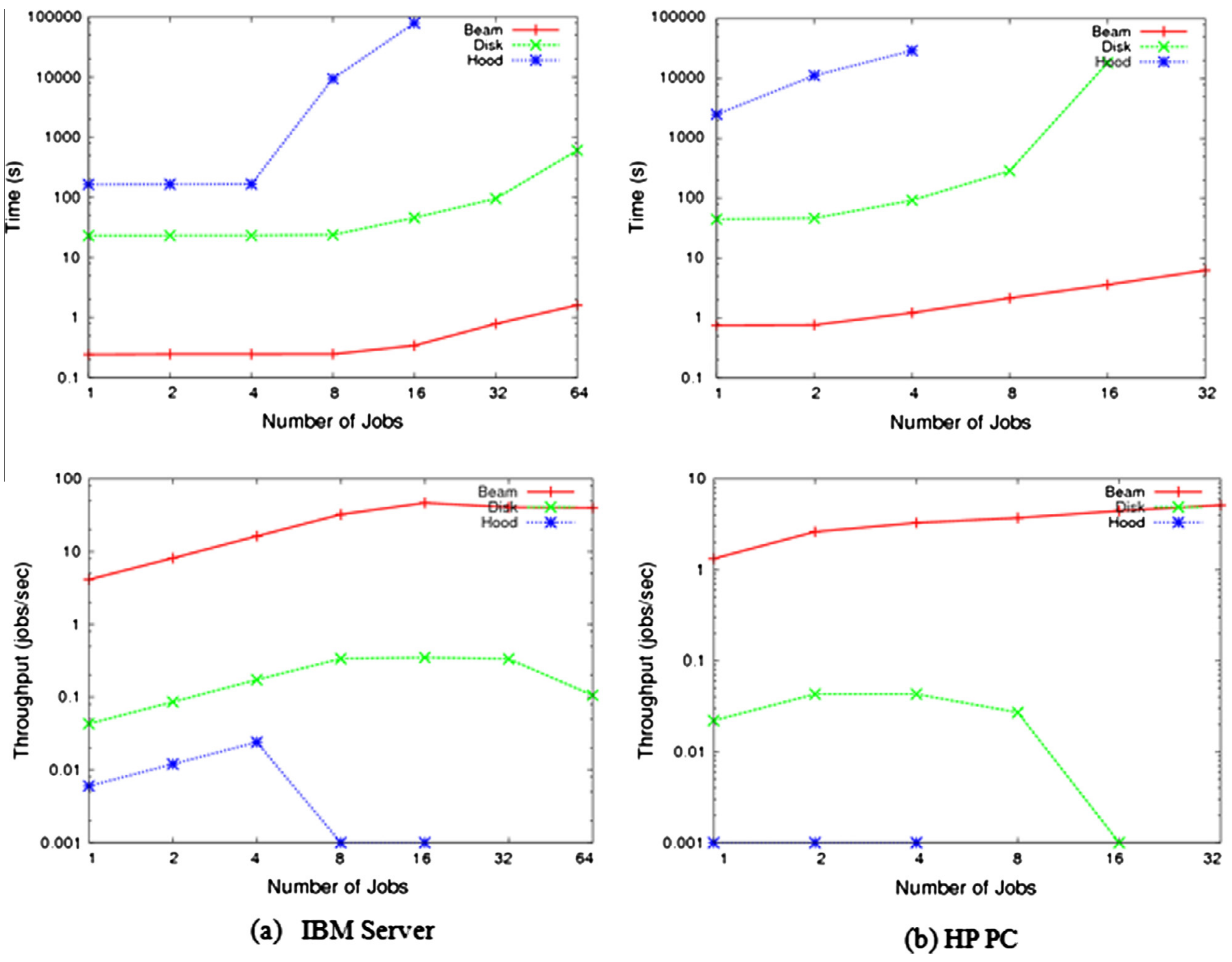


**Fig. 6.** Processing of Beam, Disk and Hood jobs with only 1 thread/job on (a) IBM server and (b) HP PC systems. The time increases and throughput drops sharply when the total memory capacity and/or core count is exceeded. Otherwise if there is enough memory, the processing time will increase linearly with the job count/core count.

system is a high-performance IBM Blade system in our data center with 4 × IBM H22 Blade Servers 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,[3] and two 72 GB 15000 rpm disks each. To summarize, PCs have 2 cores and servers have 8 cores per node. We installed RedHat Enterprise Linux 5 server OS, CalculiX, SPOOLES, and openMPI on all of these systems. We use the four FEA workloads described in the previous section for performance comparisons, namely Beam, Disk, Hood and Brake and their mixes.

### 4.2. Single S/W thread and OS scheduler

In this section, we run a controlled experiment by running $J$ identical jobs of the three workloads (Beam, Disk, Hood) concurrently on a single server node and a single PC node. We set the software multi-threading (MT) parameter as $T = 1$.

Fig. 6 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. We do not see any latency increase per increasing job count until we reach the physical core count, C. In the sustained performance (i.e. flat latency) regions we found 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, we are able 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 we will see there may be other factors affecting the performance.

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. We did memory profiling using Valgrind Massif profiler [15] and found 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. We investigated the subject with different multi-threading parameters, linear-vs.-non-linear analysis and with objects that have larger NZE (e.g. Brake) in Section 4.5.

[3] Some analyses were done when our servers had 12 GB RAM. Later they were upgraded to 24 GB.

Fig. 7 shows the dynamically-changing memory allocation of CalculiX for the Hood job over time for linear and nonlinear analysis with 1 thread.[4] 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. We find that 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, we find that 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, we repeated the same experiments with different thread counts (1–2–4–8 threads) on each model. We 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 data as this data is shared among the threads. We 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.

Referring back to Fig. 6, 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 $J \leqslant C$. We also observe a significant performance difference between the time and throughput results of the server (3–10$\times$) and the PC. We attribute this to the differences in CPUs, bus speeds, and memory sizes. We conclude that 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$\times$ more jobs completed at the same time) justifies the higher price of servers.

## 4.3. Multi S/W thread and OS scheduler

In this section, we increase the multi-threading (MT) parameter, $T$, at the application level to see whether we can gain benefits 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.

Fig. 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, we deduce that we can extend the rule $J \leqslant C$ from Section 4.2 as $J \times T \leqslant C$. 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 50 s as $16 \times 1 > C = 8$. Therefore, we again find the best latency-throughput tradeoff 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.

The results in Fig. 10 further confirm our 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. We call this the "aggressive strategy" and compare it to our "smart scheduling" in Section 4.5.

### 4.3.1. Multi-threaded nonlinear analysis

Since nonlinear analysis is iterative, we tested the effect of multi-threading 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 we have 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. We attribute the increased speedup values in nonlinear analysis 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. <0.0001%), then the total non-linear analysis time can be approximately calculated as $linear\_time \times iteration\_count \times speedup\_ratio$ between linear and nonlinear analysis. Also note that this is a property of our 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).

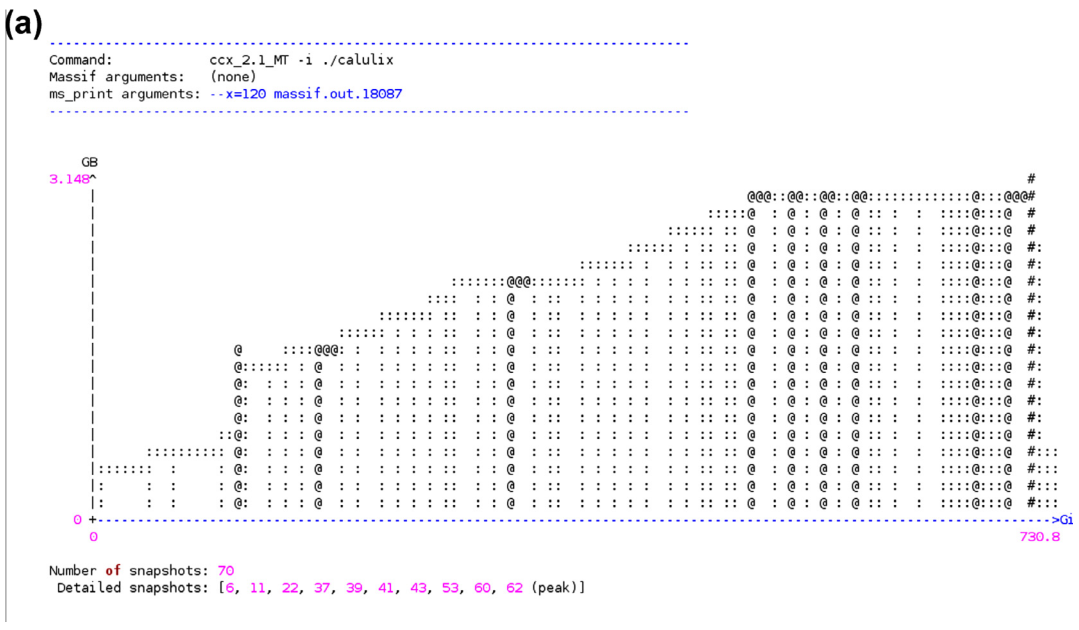
We completed our nonlinear time analysis for the Hood and Disk jobs with different concurrent job and thread counts and reported findings in Appendix B. The results further confirm our $J \times T \leqslant C$ rule.

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.

## 4.4. MT vs. MPI, single and multi-nodes

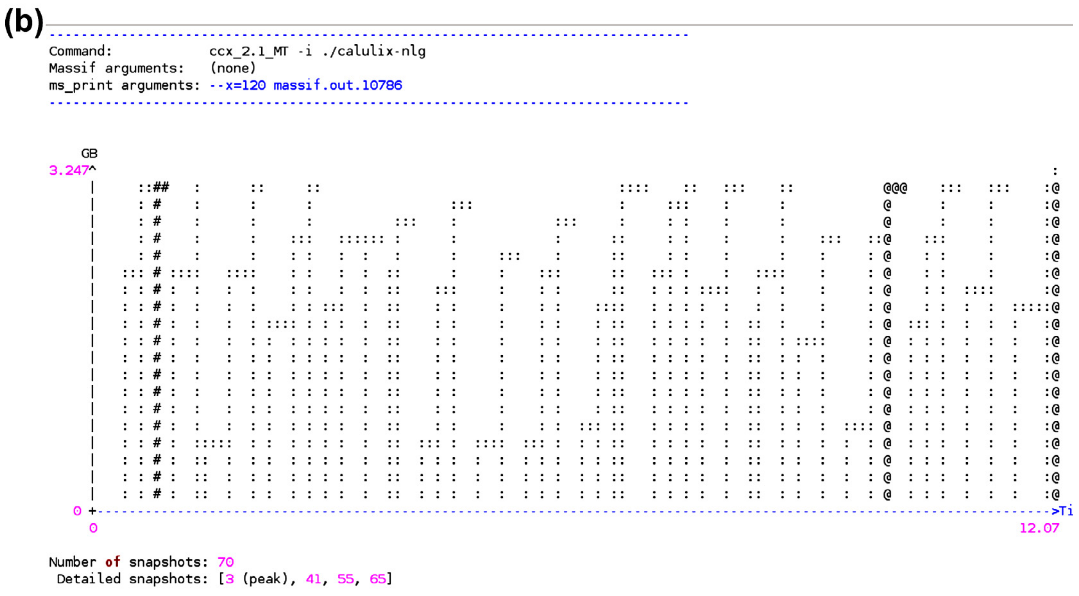
We know that MPI (distributed memory model) will communicate via explicit message passing and in MT threads will share the same process address space (shared memory model). We first quantify the overhead associated with messaging irrespective of any network equipment capabilities (i.e. in a single machine). Table 3 shows the comparison results for software multi-threading

[4] Memory profile of other objects look very similar and therefore they are not shown for brevity.



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

```
Number of snapshots: 70
 Detailed snapshots: [6, 11, 22, 37, 39, 41, 43, 53, 60, 62 (peak)]
```



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

```
Number of snapshots: 70
 Detailed snapshots: [3 (peak), 41, 55, 65]
```

**Fig. 7.** 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.

**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 |

(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 ~0.82 level (i.e. 18% performance loss) even at 8 cores.

Recent publications suggest designing new processor architectures called Message Passing Processors [16] which are distributed memory multi-core processors that communicate using message-passing among themselves. However, our finding 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, we have 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 we take MT and MPI 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 ~ 2.79× gain). Similar gains are observed for the $TC = 4$ $[2 \times 2$ and $4 \times 1]$ and $TC = 8$ $[4 \times 2$ and $8 \times 1]$ configurations over MT, but with diminishing returns as

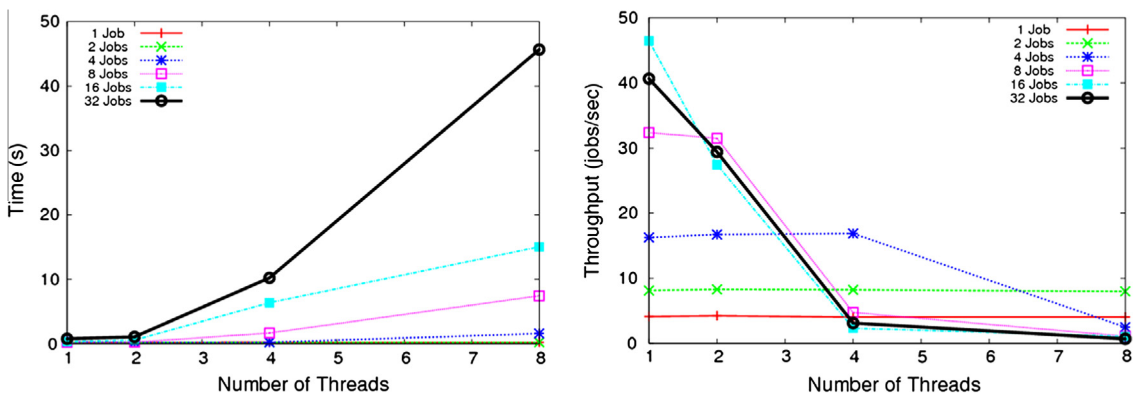


**Fig. 8.** Beam: execution time and throughput results for concurrent multi-core processing on IBM server.

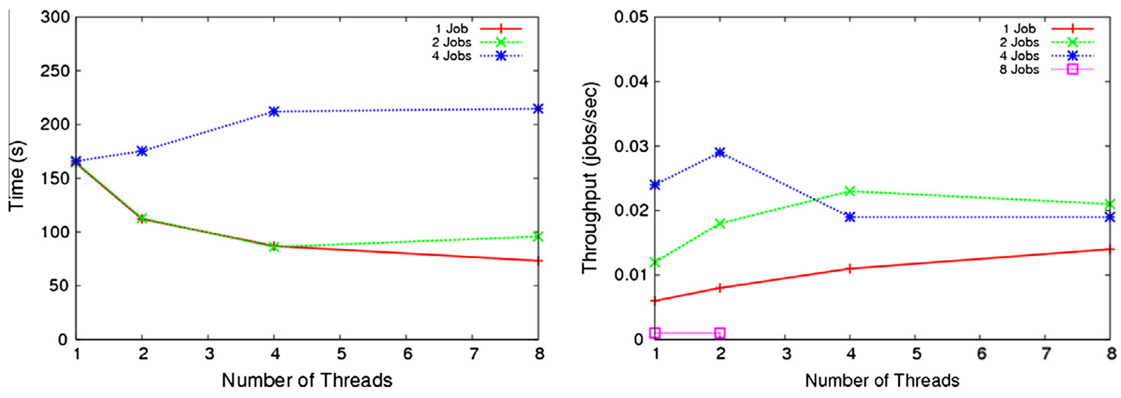


**Fig. 9.** Disk: execution time and throughput results for concurrent multi-core processing on IBM server.

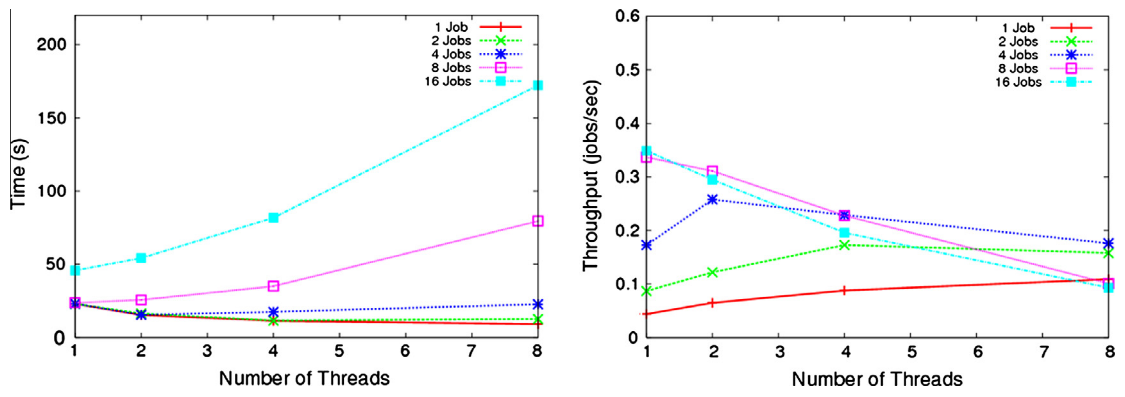


**Fig. 10.** Hood: execution time and throughput results for concurrent multi-core processing on IBM server (with 12 GB RAM).

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

| Thread | Linear | | Nonlinear | |
|---|---|---|---|---|
| | 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 job does not benefit from the additional memory resources. Note that our network bandwidth for this PC cluster was 100 Mbps and this may be the cause of some of the bottlenecks. In the future, we plan to repeat experiments with 1–10 Gbps switches for this PC cluster.

### *4.5. Smart scheduling for mixed jobs*

In previous sections, we evaluated each workload type (i.e. the Beam, Disk and Hood) extensively, but separately, on different number of cores and with varying threading (*T*) and concurrent job count (*J*) parameters. In this section, we mix these three workloads to get one step closer to a real life batch processing scenario. Given the processing limits of the systems in Fig. 6 and findings in previous subsections, we created a job mix that would substantially load the system and highlight the effects of different scheduling strategies. The mix consists of 4 Hood, 32 Disk, and 1024

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

| Core | HOOD | | | Disk | | |
|---|---|---|---|---|---|---|
| | 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 |

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

| Total core | Node × core | Time (s) |
|---|---|---|
| 2 | 1 × 2 | 1325.4 (Swap) |
| 2 | 2 × 1 | 475.5 |
| 4 | 2 × 2 | 278.5 |
| 4 | 4 × 1 | 268.2 |
| 8 | 4 × 2 | 318.0 |
| 8 | 8 × 1 | 271.0 |

**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) |

Beam jobs and the goal is to finish the batch as quickly as possible. Three different scheduling strategies we 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.

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 non-zero 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 1 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 2 servers, Smart provides a speedup of 6.79× over Aggressive and 1.53× over SJF. We also obtained estimates of execution times for this job mix by superposing the concurrent execution times of each job type using Fig. 6 and found these linear estimates to be highly accurate (within 1–10% range).

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

## *4.6. 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. [11] 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, we generated two new models in this section called circle and hollow-circle as shown in Fig. 11, 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 we increased the granularity of the meshing for the hollow-circle object 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 → 52s, 54% savings) for 8 threads and (396s → 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 pre-allocated 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. We leave it as future work, to use *bandedness*[5] of different matrices as a means for performance characterization and comparison among different workloads.

We continue our geometric analysis 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 we obtained several models called B1–B6 listed in Table 7. The NZE count was increased from 8.7 million to 47.1 million. We analyzed the time and memory usage of FEA for B1–B6. The last model caused

[5] http://en.wikipedia.org/wiki/Band_matrix.

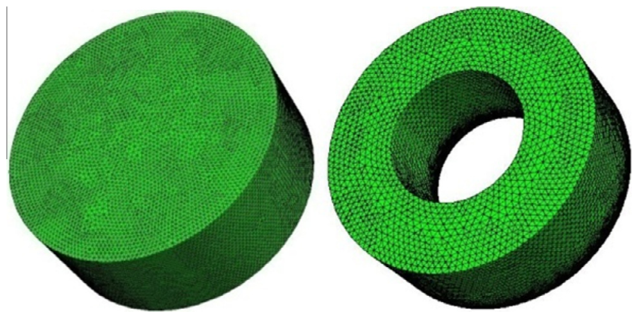


**Fig. 11.** Two control objects, circle and hollow-circle, are used to measure the effect of geometry and force on FEA processing time and memory.

**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 | 86,344 | 86,937 |
| Element | 461,234 (C3D4) | 439,169 (C3D4) |

**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 elts (million) | Memory (%) (for 24 GB) | 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 | 1.1 | 47.1 | >100 | – (>24 GB) | SWAP | SWAP |

memory swaps even on the 24 GB servers, therefore we did not increase the granularity of meshing any 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 we conclude 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.

## 5. 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.

### *5.1. 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 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). We currently make use of only performance related dimensions in this paper and our future work includes integrating other dimensions into the decision process.

### *5.2. 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 $\mathbf{E: R \rightarrow S}$. Additively homomorphic means: $E(a+b) = PLUS(E(a), E(b))$ and multiplicatively homomorphic means: $E(a * b) = MULT(E(a), E(b))$ where E is given by the function $y = E(x) = a \bmod n$, where $a = x + r \cdot p$ and the decryption is given by $x = D(y) = y \bmod p$. We tested homomorphic encryption with basic matrices and found that vector operations (additions and multiplications) can be successfully completed over encrypted data, which is the basis for FEA. However, we did not integrate this strategy 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, we believe the same techniques could be applied for privacy-preserving FEA in the cloud as well. This topic requires further investigation and constitutes our 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 [21] 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. We are currently considering the use of ABAC in our cloud service.

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 we do not get into details here, but refer users back to publications on data redundancy and system fault-tolerance. 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 effects of) these types of attacks.

### 5.3. Pricing and accounting

Cloud service providers and telecommunication companies have adopted several models for pricing and charging. 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 we need to monitor the server CPU and memory usage or analysis counts accurately. We are currently testing the Hyperic HQ tool shown in Fig. 12 and Hyperic Sigar API for online per-process resource (CPU, memory, network) monitoring. We also have additional counters attached to the scheduler, which tracks the jobs completed per client.

### 5.4. Standardization and portability

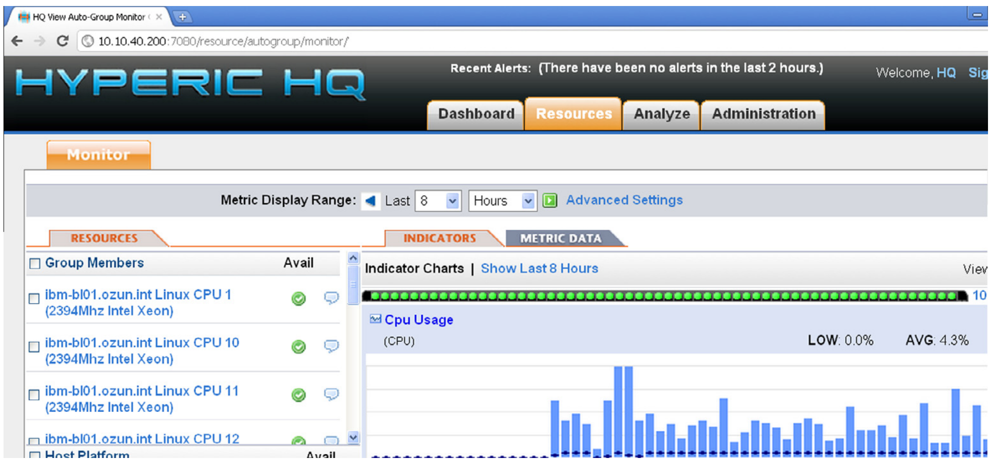
Strong progress on horizontal integration of cloud services has been accomplished and there is ongoing work on vertical integration. Standards such as Open Virtualization Format (OVF) allow IaaS providers to export and import Virtual Machines, which could be created at other provider sites. This way 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) [17] also focused their initial efforts of IaaS interoperability, but over time evolved to include PaaS and SaaS layer integration issues. Our 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, we will expand our list of supported input, output file formats 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, we believe replicating and provisioning cloud FEA services in different parts of the world may still be a valuable proposition.

## 6. Related and future work

During the last few years, there have been attempts to establish HPC cloud services or HPC as a service. In February 2010, SGI announced Cyclone–HPC Cloud [18]. Cyclone hosts both commercial and open-source software solutions for several disciplines including computational biology and chemistry, fluid dynamics, FEA, and electromagnetic. The licensing issues for the commercial software are left to the clients. Cyclone provides an IaaS model that lets clients install and run their software on SGI's hardware either on dedicated or multi-tenant servers.

Penguin Computing also provides an HPC service called Penguin Computing on Demand (POD). POD also serves at the IaaS level and makes CPU, GPU, and storage resources available through the Internet for technical computing. 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 [19]. VM images with preinstalled CalculiX solvers would be used to solve the problem on Sun's infrastructure service at 1$/CPU-h rate. The service is currently not available. Recent research also suggests that IaaS-based HPC cloud



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 | ✓ |
| Host Platform | Avail |

LOW: 0.0% AVG: 4.3%

Fig. 12. Overall (or per process) CPU, memory network and other resource usages can be tracked with Hyperic for cloud service accounting.

**Table B1**
Processing time of Hood nonlinear FEA with different concurrency and threading parameters.

| Threads | CPU processing time (s) – job counts | | | |
|---|---|---|---|---|
| | 1 | 2 | 4 | 8 |
| 1 | 2517.218 | 2530.511 | 2747.778 | *2612.259* |
| 2 | 1833.705 | 1793.619 | *1731.564* | 5116.435 |
| 4 | 1220.27 | *1252.984* | 1823.222 | 5613.863 |
| 8 | *983.537* | 1333.309 | 2316.989 | 9250.214 |
| 16 | 1057.107 | 1802.272 | 3427.119 | 7880.564 |

**Table B2**
Processing time of Disk nonlinear FEA with different concurrency and threading parameters.

| Threads | CPU processing time (s) – job counts | | | |
|---|---|---|---|---|
| | 1 | 2 | 4 | 8 |
| 1 | 43.801 | 44.134 | 44.576 | *47.599* |
| 2 | 28.541 | 28.779 | *29.25* | 51.309 |
| 4 | 21.129 | *21.79* | 33.481 | 63.363 |
| 8 | *16.884* | 23.305 | 51.142 | 198.644 |
| 16 | 19.433 | 98.624 | 221.27 | 467.365 |

services [31,32] 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. Similar to our 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 we do in this paper. 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.

There are hundreds of supercomputers in the world [20] and some of these are providing FEA computation services along with other HPC services to their communities. This service model could be considered a community-based cloud service model, which is not open to the use of general public. We wish to reach broader communities, SMBs and individuals with our public service model.

Scheduling, especially production scheduling, is also a major research field inside Industrial Engineering. Chiang et al. [22] address the Flexible Manufacturing Scheduling (FMS) problem with Critical Ratio-Based heuristics and genetic algorithm. Other algorithms called dispatching rules [25] include FIFO, Shortest Processing Time (SPT) [30], Critical Ratio (CR), EDD (Earliest Due Date), and Shortest Remaining Processing Time (SRPT). Park et al. [30] 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. Our work differs from [30] and other prior scheduling work such that we can split the FEA tasks further and choose among MP vs. MPI parallelization models before scheduling tasks onto machines. Therefore, we are neither bound to nor depend on the performance of other heuristics. In fact, we show in Table 5 that we outperform both SJF and Aggressive schedulers. Doulamis et al. [26] 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 [24] examine the computability of nonlinear problems in solid and structural mechanics problems. A measure of the level of difficulty ($L$: $1 \to 10$) is proposed and examples of typical engineering simulations are classified by this measure. In the future, we will compare our smart scheduler 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 future work. Amoura et al. [23] 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) [29] 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. We did not focus on analysis of assembled (multi-part) systems and computational dependencies among them in this paper.

To summarize, our future work consists of extending the service into areas shown in Fig. 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 our FEA service.

## 7. Conclusions

In this paper, we described the design and implementation of our Finite Element Analysis cloud service with a focus on mechanical structural analysis, performance characterization and job scheduling issues. We characterized the CPU and memory requirements of representative structural mechanics workloads and addressed some of the performance challenges related to concurrent job processing. We did an extensive performance benchmarking of linear and nonlinear jobs over multi-core and multi-node computing resources. We showed 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.

We hope our service will simplify 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.

## Acknowledgements

This work has been sponsored in part by the European Union FP7 Marie Curie program under BI4MASSES project and IBM Shared University Research (SUR) program. Dr. Ismail Ari is also supported by the TUBITAK (Turkish National Science Foundation) Career Award 109E194 and Nitel Muhtaroglu is supported by the IBM PhD Fellowship award. We would like to thank all our sponsors and other members of the Cloud Computing Research Group at Ozyegin University.

## Appendix A. Details in CalculiX

Several critical features of our system depend on the open-source FEM tool called CalculiX (http://www.calculix.de) and the

SPOOLES linear equation solver. We described some of the relevant features and configuration parameters throughout the paper. Other capabilities of CalculiX can be summarized as follows.

### *A.1. Types of analyses supported*

Static analysis, frequency analysis, buckling analysis, modal dynamic analysis, steady state dynamics, direct integration dynamic analysis, heat transfer, acoustics, shallow water motion, hydrodynamic lubrication, irrotational incompressible inviscid flow, electrostatics, stationary groundwater flow, diffusion mass transfer in a stationary medium, aerodynamic networks, hydraulic networks, turbulent flow in open channels, 3D Navier–Stokes calculations.

### *A.2. Types of materials supported*

CalculiX also supports analysis with nonlinear materials including elastic materials, hyperelastic and hyperfoam materials, deformation plasticity, incremental viscoplasticity, and user defined materials. Ricks discusses the application of Newton's method to the problem of elastic stability [28].

## Appendix B. Additional nonlinear analysis results

The nonlinear analysis of Hood and Disk jobs on IBM server (8 core, 24 GB RAM) with different threading parameters are given in Tables B1 and B2, respectively. The italicized results show the minimum CPU processing time for each job–thread configuration. We see that the $J \times T \leqslant C$ rule holds for nonlinear analysis as well (i.e. $8 \times 1$, $4 \times 2$, $2 \times 4$, $1 \times 8$ being the shortest time runs).

## References

[1] Mell P, Grance T. The NIST definition of cloud computing draft, 800-145, January 2011. <http://www.nist.gov/itl/cloud/index.cfm> [last accessed 07.06.12].
[2] Amazon web services. <http://aws.amazon.com>.
[3] Dassault systemes product lifecycle management (PLM) solutions and CATIA design. <http://www.3ds.com/>.
[4] Bremberg D, Dhondt G. Automatic crack-insertion for arbitrary crack growth. Eng Fract Mech 2008;7(3–4):404–16.
[5] Schöberl J. NETGEN: an advancing front 2D/3D-mesh generator based on abstract rules. Comput Visual Sci 1997;1(1):41–52.
[6] Web App framework. <http://en.wikipedia.org/wiki/List_of_web_application_frameworks>.
[7] Dhondt G. CalculiX CrunchiX User's manual v2.0. <http://www.calculix.de> and <http://www.bconverged.com>.
[8] Wittig K. CalculiX user's manual–CalculiX GraphiX, V2.0. <http://www.bconverged.com/calculix/doc/cgx/>.
[9] WebGL by the Kronos Group. <http://www.khronos.org/webgl/>.
[10] Ashcraft C, Pierce D, Wah DK, Wu J. 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/>.
[11] Hsieh SH, Yang YS, Hsu PY. Integration of general sparse matrix and parallel computing technologies for large-scale structural analysis. Comput Aid Civil Infrastruct Eng, USA 2002;17(6):423–38.
[12] Dean J, Ghemawat S. MapReduce. Simplified data processing on large clusters, Usenix, OSDI 2004. p. 137–50.
[13] MATLAB sparse matrix operations. <http://www.mathworks.com/help/techdoc/math/>.
[14] Amdahl's law speedup chart. <http://en.wikipedia.org/wiki/Amdahl's_law>.
[15] Seward J, Nethercote N, Weidendorfer J. Valgrind 3.3-advanced debugging and profiling for GNU/Linux applications, 2008. ISBN: 0954612051 9780954612054.
[16] Howard J, et al. A 48-Core IA-32 message-passing processor with DVFS in 45nm CMOS. In: Proceedings of ISSCC 2010 (IEEE international solid-state circuits conference), February 2010.
[17] Open cloud computing interface standard by open grid forum OCCI working group. <http://occi-wg.org/>.
[18] Joseph EC, Conway S, Wu J. A new approach to HPC public clouds: the SGI cyclone HPC cloud. <http://www.sgi.com/pdfs/4215.pdf>.
[19] Sun Grid. <http://www.sun.com/service/sungrid/>.
[20] TOP500 Super-computers list. <http://www.top500.org>.
[21] Li J, Ari I, Jain J, Karp AH, Dekhil M. Mobile in-store personalized services. In: 7th IEEE international conference on web services (ICWS), 2009.
[22] Chiang Tsung-Che, Fu Li-Chen. Solving the FMS scheduling problem by critical ratio-based heuristics and the genetic algorithm. In: IEEE international conference on robotics and automation (ICRA), vol. 3, April–May 2004. p. 3131–6.
[23] Amoura, Bampis, Kenyon, Manoussakis. Scheduling independent multiprocessor tasks. Algorithmica 2002;32(2):247–61.
[24] Belytschko T, Mish K. Computability in nonlinear solid mechanics. Int J Numer Methods Eng 2001;52:3–21.
[25] Chik, M.A., Ahmad, I., Jamaluddin, M.Y. A simulation approach for dispatching techniques comparison in 200mm wafer foundry. In: IEEE international conference on semiconductor electronics (ICSE), 2004. p. 645–9.
[26] Doulamis N, Doulamis A, Litke A, Panagakis A, Varvarigou T, Varvarigos E. Adjusted fair scheduling and nonlinear workload prediction for QoS guarantees in grid computing. Comput Commun 2007;30(3):499–515 [special issue: Emerging middleware for next generation networks].
[27] Tsap LV, Goldgof DB, Sarkar S, Huang W-C. Efficient nonlinear finite element modeling of nonrigid objects via optimization of mesh models. Comput Vis Image Understand 1998;69(3):330–50.
[28] Ricks E. The application of Newton's method to the problem of elastic stability. J Appl Mech 1972;39(4):1060–5.
[29] Jörg Dümmler, Thomas Rauber, Gudula Rünger. Scalable computing with parallel tasks. In: ACM Proceedings of the 2nd workshop on many-task computing on grids and supercomputers (MTAGS), 2009.
[30] Park Y, Casey K. A novel adaptive support vector machine based task scheduling, parallel and distributed computing, 2010.
[31] Vallee G, Naughton T, Engelmann C, Hong Ong, Scott SL. System-level virtualization for high performance computing. In: 16th Euromicro conference on parallel, distributed and network-based processing, February 2008. p. 636–43.
[32] Xiaoyong B. High performance computing for finite element in the cloud. In: IEEE international conference on future computer sciences and application, 2011.

Dr. Ismail Ari received his PhD degree in 2004 from Computer Science Department of University of California, Santa Cruz. Between 2004-2009 he worked as a researcher in Hewlett Packard Labs in Silicon Valley, USA. His research interests include cloud computing, service-oriented architectures, data mining, and networked storage systems. He has international publications and US patents related to these topics. After joining Özyeğin University in 2009 he has received several awards and research grants including IBM Top Faculty Contributor Award, EU Marie Curie International Reintegration Grant, and TÜBİTAK (Turkish NSF) National Young Researcher Career Award. Dr. Ari is a member of IEEE and ACM.

Nitel Muhtaroglu is a Ph.D. student at Ozyegin University studying under supervision of Dr. Ari. 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. His research interests mainly focus on Computational Mechanics, Engineering Informatics, High Performance Computing and Applied Mathematics.