Exploring Application and Infrastructure Adaptation on Hybrid Grid-Cloud Infrastructure

Hyunjoo Kim[1], Yaakoub el-Khamra[3,4], Shantenu Jha[2], Manish Parashar[1]

[1] *NSF Center for Autonomic Computing, Dept. of Electrical & Computer Engr., Rutgers University, , NJ, USA*

[2] *Center for Computation & Tech. and Dept. of Computer Science, Louisiana State University, USA*

[3] *Texas Advanced Computing Center, The University of Texas at Austin, Austin Texas, USA*

[4] *Craft and Hawkins Dept. of Petroleum Engineering, Louisiana State University, USA*

Abstract

Clouds are emerging as an important class of distributed computational resources and are quickly becoming an integral part of production computational infrastructures. An important but oft-neglected question is, what new applications and application capabilities can be supported by clouds as part of a hybrid computational platform? In this paper we use the ensemble Kalman-filter based dynamic application workflow and investigate how clouds can be effectively used as an accelerator to address changing computational requirements as well as changing Quality of Service constraints (e.g., deadlines). Furthermore, we explore how application and system-level adaptivity can be used to improve application performance and achieve a more effective utilization of the hybrid platform. Specifically, we adapt the ensemble Kalman-filter based application formulation (serial versus parallel, different solvers etc.) so as to execute efficiently on a range of different infrastructure (from High Performance Computing grids to clouds that support single core and many-core virtual machines). Our results show that there are performance advantages to be had by supporting application and infrastructure level adaptivity. In general, we find that grid-cloud infrastructure can support novel usage modes, such as deadline-driven scheduling, for applications with tunable characteristics that can adapt to varying resource types.

Categories and Subject Descriptors D.1.3 [Concurrent Programming]: Distributed Programming, Parallel Programming **General Terms** Algorithms, Management, Performance, Experimentation **Keywords** Cloud Computing, Hybrid Computing

HPDC'10, June 20–25, 2010, Chicago, Illinois, USA.

I. Introduction and Motivation

Clouds are emerging as an important class of distributed computational resources – both, for data-intensive and compute-intensive applications. In fact, they are rapidly joining high-performance grids as viable computational platforms for scientific exploration and discovery, and it is clear that future production computational infrastructures will integrate both paradigms. It is thus critical to understand what role clouds will play in the formulation and execution of computational applications and what abstractions and tools are necessary to support their usage.

Clouds support a different although complementary usage model to more traditional High Performance Computing (HPC) grids. Cloud usage models are based upon on-demand access to computing utilities, an abstraction of unlimited computing resources, and a usage-based "payment model" whereby users essentially "rent" virtual resources and "pay" for what they use. Several recent efforts [1], [2], [3] have clearly demonstrated that clouds can be effectively used as alternate platforms for certain classes of applications. These include applications with very modest communication and synchronization requirements such as parameter sweeps and data analytics. Many applications that currently use clouds are legacy or cross-over applications from the cluster/HPC world. For example, there are many traditional HPC applications that use clouds such as Cyclone [4] – SGI's latest cloud offering. Furthermore, it has also been established that there are application profiles that are better suited to HPC grids (e.g., large scale DNS and Quadratic Mean calculations), and others that are more appropriate for clouds.

Whereas it is important to explore and support the migration of traditional applications to cloud computational platforms, it is also imperative to ask: What new applications and application capabilities can be supported by clouds – either as stand-alone infrastructure, or as part of a hybrid grid-cloud computational platform? Can the addition of clouds enable scientific applications and usage

modes, that are not possible otherwise? What abstractions and systems are essential to support these advanced applications on different hybrid grid-cloud platforms (e.g., High-Performance Computing (HPC) and/or High Throughput Computing (HTC) grid-clouds). These questions and related issues frame the scope of this paper.

We address these issues in the context of dynamic applications, defined as applications whose execution trajectory or resource requirements cannot be completely determined a priori. This can be due to a change in computational requirement or a change in execution trajectory. Furthermore, an application is referred to as dynamic only if it is able to respond to changes. Ideally, dynamic applications are able to adapt at the application level, as well as at the infrastructure utilization level. Thanks to the ability to provide an illusion of unlimited and/or *immediately available* resources, as currently provisioned, clouds have the interesting and useful ability to support applications with changing requirements. Furthermore, dynamic applications due to their ability to adapt to changes in infrastructure can effectively utilize clouds resources when necessary.

We established in Ref. [5] that clouds in conjunction with traditional HPC and HTC grids provide a balanced infrastructure supporting scale-out and scale-up/down for a range of application (model) sizes and requirements. We also demonstrated that such hybrid infrastructure support a range of objectives, for example, reduced times-to-solutions, reduced costs (in terms of currency or resource usage), or resilience against unexpected outages (e.g., unexpected delays in scheduling queues or unexpected failures). In this paper, we investigate how clouds can be effectively used as accelerators to address changing computational requirements as well as changing Quality of Service (QoS) constraints (e.g., deadline) for a dynamic application workflow. Furthermore, we explore how application and system-level adaptivity can be used to achieve improved overall application performance and more effective utilization of the hybrid platform.

Two high-level goals guide the design of the experiments presented in this paper. One goal of this paper is to establish the benefits from a hybrid HPC grid-cloud execution environment for a sophisticated scientific application. This extends the work in Ref. [5], where we now have a much richer space of infrastructure and application variability – including more virtual machine types, sequential and parallel implementations of the scientific applications. We investigate how acceleration – defined as a reduced TTC, can be achieved by exploring & exploiting a richer set of infrastructure, but we also investigate these objectives in the context of a richer application space, i.e., different solvers, pre-conditioners etc., predict and model parallel tasks and their usage of hybrid HPC grids and cloud infrastructure.

In the second goal, we want to investigate application-infrastructure adaptivity, how it can be supported on hybrid-infrastructure, and the subsequent performance advantages. Specifically, we want to validate the conceptual architecture and framework for supporting adaptivity as defined in Ref [6]. To this end, three approaches to supporting adaptivity in computational science applications are investigated: (i) *Track 1* – referred to as infrastructure-level adaptivity is characterized by the selection of infrastructure as a degree-of-freedom; (ii) *Track 2* – referred to as application-level adaptivity, and which is characterized by the tuning of applications in various ways, and (iii) *Track 3* – adaptivity of both infrastructure and application. For example, track 1 involves provisioning of immediately available resources; technically, track 2 is about changing the numerical solvers & preconditioners, and the number of ensemble members, size of ensembles and possibly also the frequency of assimilation. A necessary condition is that the application formulation should be amenable to and support runtime adaptivity. Track 3 considers both infrastructure adaptivity and application tuning at the same time. We aim to understand the relative merits and performance trade-offs of the three tracks.

Our adaptive workflow is an ensemble Kalman-filter (EnKF) that uses a reservoir simulator [7] to forecast an ensemble of models and a repeating analysis step to match model production with historical production (history matching). We investigate the application along the three tracks described above. Experimental results show that adaptivity can reduce total time-to-completion (TTC) with a reasonable cost. Experiments also establish that application-level adaptivity can affect TTC.

The rest of this paper is organized as follows. Section II discusses related works and provides some background information. Section III describes reservoir characterization and the EnKF applications workflow, introduces CometCloud, and then describes dynamic execution and adaptivity of EnKF using CometCloud. In Section IV we describe the experimental environments and present our results. Section VI concludes this paper.

II. Related and Prior Work

A. Related Work

In Ref. [8], Buyya et.al. describe an approach of extending a local cluster to cloud resources using schedulers applying different strategies. Their approach was simulated-based, and not executed on production runs on cloud resources. References [2] and [9], investigate the elastic growth of grids to clouds. Vazquez et.al. [9] use the GridWay metascheduler to elastically grow the grid infrastructure to Nimbus [10] based cloud. Their experimented used the NAS Grid Benchmark suite. Ostermann et.al. [2] extended a workflow application developed for a grid computing environment to include cloud resources; they used

the Austrian Grid and an Eucalyptus-based academic cloud installation.

Our previous work considered a hybrid computing environment which integrated clusters with clouds [11], grids with clouds [5] and enabled autonomic cloudbursts on demand. CometCloud is very flexible and able to integrate multiple resource types, while other approaches are closely based on one resource class, for example, they use some specific grid based job scheduler, resource management, etc. CometCloud uses an overlay mechanism, therefore any kind of node can join CometCloud and provide computing or/and storage capability.

Several economic models for resource scheduling on grids have been proposed, however, there exist limited effort on resource scheduling on hybrid computing environments. Recently a combinatorial auction model [12] was proposed for both grids and clouds and a cost model based on economic scheduling heuristics [13] was investigated for cloud-based streams. An adaptive scheduling mechanism [14] used economic tools such as market, bidding, pricing, etc. on an elastic grid utilizing virtual nodes from clouds. Other tools and resource management systems used include GridARM [15] and GLARE [16].

In Ref. [17], on-demand resource provisioning mechanism based upon load was presented. In contrast, resource provisioning in this paper is based on user objective and metrics. The autonomic scheduler decides on the mix of resource classes and the number of nodes depending on user-defined objectives, the estimated runtime of tasks and the cost calculated from the time-to-completion.

B. Objective Driven Hybrid Usage of HPC Grids and Clouds

Developing and deploying applications on a hybrid and dynamic computational infrastructure presents new and interesting challenges. There is the need for programming systems that can express the hybrid usage modes and associated runtime trade-offs and adaptations, as well as coordination and management infrastructures that can implement them in an efficient and scalable manner. Key issues include decomposing applications, components and workflows, determining and provisioning the appropriate mix of grids/clouds resources, and dynamically scheduling them across the hybrid execution environment while satisfying/balancing multiple, possibly changing objectives for performance, resilience, budgets and so on [18].

In Ref. [5], we investigated the integration of HPC grids and clouds and how an autonomic framework was used to support the following autonomic objectives:

- **Acceleration:** This use case explores how clouds can be used as accelerators to improve the application time-to-completion by, for example, using cloud resources to alleviate the impact of queue wait times or exploit an additionally level of parallelism by offloading appropriate tasks to cloud resources, given appropriate budget constraints.
- **Conservation:** This use case investigates how clouds can be used to conserve HPC grid allocations, given appropriate runtime and budget constraints.
- **Resilience:** This use case will investigate how clouds can be used to handle unexpected situations such as an unanticipated HPC grid downtime, inadequate allocations or unanticipated queue delays.

C. Conceptual Architectures for Adaptivity in Computational Science

Looking at existing practices in computational science, two corresponding conceptual architectures can be observed. As discussed in Ref. [19], both of these architectures are composed of the application, a resource manager that allocates, configures and tunes resources for the application, and an autonomic manager that performs the tuning of application and/or system parameters.

In the first conceptual architecture, the application and resources are characterized using a number of *dynamically modifiable* parameters/variables that have an impact on the overall *observed behavior* of the application. Each of these parameters has an associated range over which it can be modified, and these constraints are known *a priori*. The intention of the autonomic tuning manager is to alter these parameters based on some overall *required behavior* (referred to as the application objective) that has been defined by the user. Tuning in this case is achieved by taking into account, for example, (i) historical data about previous runs of the application on known resources, obtained using monitoring probes on resources; (ii) historical data about previous selected values of the tunable parameters; (iii) empirically derived models of application behavior; (iv) the specified tuning mechanism and strategy; etc. For example, an autonomic tuning mechanism in this architecture may involve changing the *size* of the application (for instance, the number of data partitions generated from a large data set, the number of tiles from an image, etc.), or the set of parameters over which execution is being requested. This tuning is used to make desired tradeoffs between quality of solution, resource requirements and execution time or to ensure that a particular QoS constraint, such as execution time, is satisfied.

Another conceptual architecture, where the application is responsible for driving the tuning of parameters, and choosing a tuning strategy. The autonomic tuning manager is now responsible for obtaining monitoring data from resource probes and the strategy specification (for one or more objectives to be realized) from the application. Tuning now involves choosing a resource management strategy that can satisfy the objectives identified by the application. An

example of such an architectural approach is the use of resource reservation to achieve a particular QoS requirement. The G-QoSM framework [20] demonstrates the use of such an architecture, involving the use of a soft real-time scheduler (DSRT) along with a bandwidth broker to make resource reservation over local compute, disk and network capacity, in order to achieve particular application QoS constraints.

We conclude this sub-section with some observations: A key underlying concept is the separation of management and optimization policies from enabling mechanisms that allows a repertoire of a mechanisms to be automatically orchestrated at runtime to respond to the heterogeneity and dynamics, both of the applications and the infrastructure. Examples of mechanisms could be alternative numerical algorithms, domain decomposition, and communication protocols. If this were an autonomics paper, we would focus on developing policies and strategies that are capable of identifying and characterizing patterns at design and at runtime and, using relevant policies, to manage and optimize the patterns. However, in this work we will not focus on the policies and metrics that determine adaptivity, but will focus on the capabilities and performance arising as a consequence of the adaptivity.

We reiterate that the conceptual architectures – tuning by and of applications, are not exhaustive [19], but provide an initial formulation with a view towards understanding adaptivity in the EnKF application that we investigate. As a final observation, we note that application and system specific runtime adaptations are widely used in computational science applications as a means for managing applications and tuning performance. But supporting them on production grade infrastructure and at scale has been challenging.

III. Application Characterization and Dynamic Execution

A. Reservoir Characterization: EnKF-based History Matching

EnKF represent a promising approach to history matching [21], [22], [23], [24]. EnKF are recursive filters that can be used to handle large, noisy data; the data in this case are the results and parameters from an ensemble of reservoir models that are sent through the filter to obtain the "true state" of the data. Since the model varies from one ensemble member to another, the run-time characteristics of the ensemble simulation are irregular and hard to predict. Furthermore, during simulations, when real historical data is available, all the data from the different ensemble members at that simulation time must be compared to the actual production data before the simulations are allowed to proceed. This translates into a global synchronization point for all ensemble members in any given stage.

The variation in computational requirements between individual tasks and stages can be large. As a result, efficient execution of large scale complex reservoir characterization studies requires dynamic runtime management to ensure effective resource utilization and load-balancing. Furthermore, since the computational requirements are no know a priori, the application can potentially benefit from the elasticity of cloud resources. For this reason, performing large scale complex reservoir characterization studies can benefit greatly from the use of a wide range of distributed, high-performance and throughput as well as on-demand computing resources.

B. CometCloud Overview

CometCloud [25] is an autonomic computing engine for clouds and grids environments that enables the development and execution of dynamic application workflows in heterogeneous and dynamic clouds/grids infrastructures. It supports the on-demand bridging of public/private clouds and grids as well as autonomic cloudbursts. Conceptually, CometCloud is composed of a programming layer, service layer, and infrastructure layer. The infrastructure layer uses the Chord self-organizing overlay [26], and the Squid [27] information discovery and content-based routing substrate build on top of Chord. The service layer provides a Linda-like [28] tuple space. The programming layer provides the basic framework for application development and management including the master/worker/BOT, workflow and MapReduce/Hadoop [29], [30].

Key components of CometCloud include:

Workflow Manager: The workflow manager is responsible for coordinating the execution of the overall application workflow, based on user-defined polices.

Estimators: Estimators provide an estimate for the computational cost of each task. This estimate can be obtained through a computational complexity model or through quick, representative benchmarks.

Autonomic Scheduler: The autonomic scheduler uses the estimator to compute anticipated runtimes for tasks on available resource classes, and to determine the initial hybrid mix HPC grids/clouds resources based on user/system-defined objectives, policies and constraints.

Grids/Clouds Agents: The grid/cloud agents are responsible for provisioning the resources on their specific platforms, configuring *workers* as execution agents on these resources, and appropriately assigning tasks to these workers. CometCloud primarily supports a *pull-based* model for task allocation, where workers directly pull tasks from the Comet space. However, on typical HPC grid resources with a batch queuing system, a combined *push-pull* model is used, where we insert "pilot-jobs" [31] containing workers into the batch queues and the workers then pull tasks from the Comet space when they are scheduled to run.

At each stage of the workflow, the workflow manager determines the number of ensemble members at the stage as well as relative computational complexity of each member and then it encapsulates each ensemble member as a task. Once the tasks to be scheduled within a stage have been identified, the autonomic scheduler analyzes the tasks and their complexities to determine the appropriate mix of TeraGrid (TG) and EC2 resources that should be provisioned. This is achieved by (1) clustering tasks based on their complexities to generate blocks of tasks for scheduling, (2) estimating the runtime of each block on the available resources using the cost estimator service and (3) determining the allocations as well as scheduling policies for the TG and EC2 based on runtime estimates as well as overall objectives and resource specific policies and constraints (e.g., budgets). More details are described in [5].

C. Dynamic Execution and Adaptivity of EnKF using CometCloud

As stated earlier, we consider two types of adaptivity in this paper. Infrastructure adaptivity explores a richer infrastructure space and selects appropriate numbers and types (e.g., number and type of virtual machines), of resources based on application requirements and overall constraints. The second type of adaptivity is application adaptivity which involves adapting the structure and behavior of the applications based on application/system characteristics (e.g., the size of ensemble members, problem size and application configuration) and runtime state.

Infrastructure adaptivity is achieved by estimating each ensemble member's runtime on available resources and selecting the most appropriate resources for them. To estimate runtime on each different resource class, the CometCloud autonomic scheduler asks a worker per resource class to run the runtime estimation module, which is achieved by inserting a runtime estimation (benchmark) task into the Comet space. A worker running on each resource class pulls the task, executes it and returns the estimated runtime back to the scheduler.

If there is no active worker on a resource class, the scheduler launches a new worker on the resource class. The overhead of running an estimation task itself is 5% of that of an actual task. However, if the scheduler should start a new worker for estimation, it can cause additional time overhead, for example, the overhead of launching a new EC2 instance, or the waiting time in a queue after submitting a pilot job for TG. This runtime estimation is accomplished at the beginning of every stage because each stage is heterogeneous and the runtime of the previous stage can not be used for the next stage. Once the autonomic scheduler gathers estimated runtimes from all resource classes, it maps the ensemble members (encapsulated as asks) to the most appropriate available resource class based

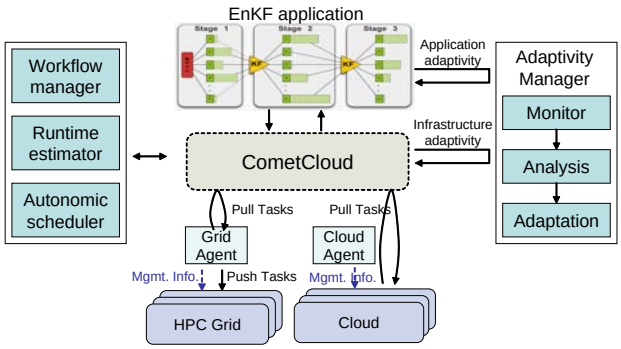


Fig. 1: Autonomic architecture for adaptivity. Workflow manager, runtime estimator, autonomic scheduler as well as adaptivity manager collaborate to reach a decision regarding the best resource provisioning and application configuration.

on the defined policy. Policies determine whether runs are made with deadline-based or cost-based (i.e. with budget limits). After then the scheduler decides the number of nodes (workers) for each resource class and the appropriate mix of resources. Naturally, workers can consume more than one task and the number of workers is typically smaller than the number of tasks.

Application adaptivity on the other hand relies heavily on the application infrastructure. Since the reservoir simulator is based on PETSc [32], we have access to a wide variety of direct and iterative solvers and preconditioners. The selection of optimal solver and preconditioner combination depends on the problem (stiffness, linearity, properties of the system of equations, etc.) as well as the underlying infrastructure. Since the simulator needs to perform several iterations, the first few iterations are performed with several solver/preconditioner combinations. This "optimization" study is performed with ensemble member rank 0 only, also known as the "base–case" from which all other ensemble members are generated. The combination with the best performance (shortest wall-clock time) is then selected and passed on to the next stage to reduce simulation runtime.

The overall system architecture used in the experiments is shown in Figure 1. Every stage of the application workflow is heterogeneous, and as a result, the selection of infrastructure and application configurations for each stage can be different. At every stage, the autonomic manager collects information about both, the infrastructure and the application, and analyzes this information to decide on appropriate resources and application configuration. These decisions affect both current stages (infrastructure adaptivity) as well as subsequent stages (application adaptivity). After reaching a decision on the most efficient infrastructure/application configurations and mix of resources, resources are provisioned and "ensemble-member-workers" are executed. On the EC2, this translates to launching appropriate VMs running custom images. On the TG, ensemble-member-

workers are essentially "pilot jobs" [31] that are inserted into the queue. The workflow manager inserts tasks into CometCloud and ensemble-workers directly access CometCloud and retrieve tasks based on the enforced scheduling policy. TG workers are allowed to pull the largest tasks first, while EC2 workers pull the smallest tasks. While this policy is not optimal, it was sufficient for our study. During the execution, the workflow manager monitors the executions of the tasks to determine progress and to orchestrate the execution of the overall workflow. The autonomic scheduler also monitors the status of the resources (using the agents), and determines progress to ensure that the scheduling objectives and policies/constraints are being satisfied, and can dynamically change resources allocation if they cannot be satisfied.

It is possible that for a given user defined deadline, that the objective of finishing all tasks at or before the deadline not be met. This could be simply due to insufficient resources for the computational load as a consequence of the deadline imposed; or it could be due to autonomic scheduling efficiency. In this paper, we focus on the situation where the objective of accelerating the solution/TTC on the TG by using EC2 is successful. In addition to the efficiency of the autonomic scheduler (which we do not analyze here), the relative capacity of the TG & EC2 resources will determine the maximum value of acceleration possible for a given workload. In other words, with the addition of sufficiently large number of cloud resources – possibly of different types of clouds, any imposed deadline will be met for a given workload.

IV. Experiments

Our experiments are organized with the aim of understanding how application and infrastructure adaptivity facilitate desired objectives to be met. Specifically, we investigate how adaptivity – application or infrastructure, or possibly both in conjunction, enable lowering of the TTC, i.e., acceleration. We explore, (1) how the autonomic scheduler reduces TTC when a deadline is specified, and (2) how adaptation helps achieve the desired objective by facilitating an optimized application and/or infrastructure configuration, or via autonomic scheduling decisions when multiple types of cloud resources are available.

We use a small number of stages of the EnKF workflow with a finite difference reservoir simulation of problem size 20x20x20 gridpoints and 128 ensemble members with heterogeneous computational requirements. Our experiments are performed on the TG (specifically Ranger) and several instance types of EC2. Table I shows the EC2 instance types used for experiments. We assume that a task assigned to a TG node runs on all 16 cores for that node (for Ranger).

TG provides better performance than EC2, but is also the relatively more restricted resource – in that there are often queuing delays. Hence, we use EC2 which is available immediately at a reasonable cost to accelerate the solution of the problem. A task pulled by EC2 node runs sequentially (in case of m1.small which has a single core), or in parallel (other instance types with multiple cores) inside a single VM. To enable MPI runs on multi-core EC2 instance, we created a MPI-based image on EC2. This image included the latest versions of compilers, MPICH2, PETSc and HDF5 and was configured for performance above all else. Although we experimented with using MPI across multiple VMs as described in Section IV-C, we exclude using data from experiments involving running MPI across VMs in the analysis of understanding the effect of adaptivity; as is expected, due to communication overheads, it displays poor performance as compared to MPI within a single VM.

TABLE I: EC2 instance types used in experiments

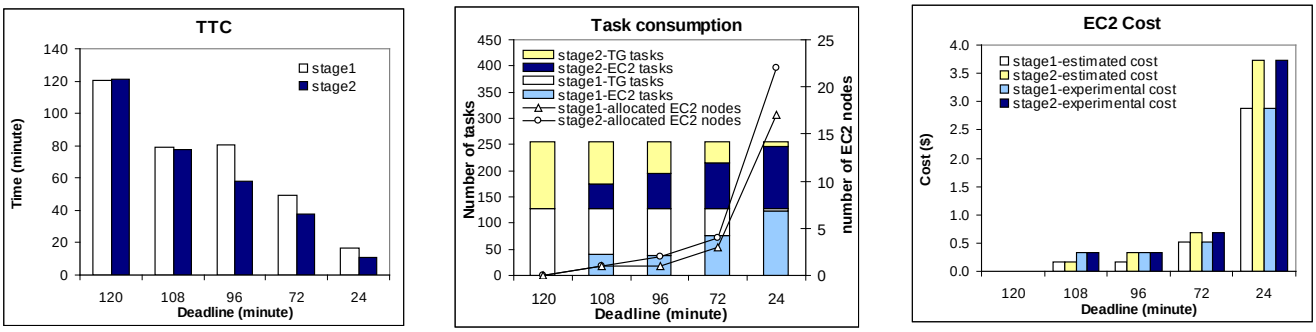
Instance type	Cores	Memory(GB)	Platform(bit)	cost($/hour)
m1.small	1	1.7	32	0.1
m1.large	2	7.5	64	0.34
m1.xlarge	4	15	64	0.68
c1.medium	2	1.7	32	0.17
c1.xlarge	8	7	64	0.68

A. Baseline: Autonomic Scheduling (Acceleration) in Response to Deadlines

When the deadline by when all tasks must be completed has been determined/decided, the autonomic scheduler estimates TTC, first assuming the availability of only TG resource. The scheduler then decides how much of the workload should be migrated to EC2 in order to meet the given deadline. If the deadline is sooner than the estimated TTC (assuming usage of only TG resources), the number of tasks which should be off-loaded onto EC2 is decided and the autonomic scheduler selects the appropriate EC2 instance types and number of such nodes. It then allocates the appropriate number of nodes of the selected instance types.

The autonomic scheduler has no knowledge of runtime characteristics of tasks on different resources, the scheduler makes decisions based on estimated runtime from the runtime estimator. The runtime estimator is a simple utility that launches a full-size ensemble member simulation with a reduced number of iterations to minimize cost. The results of various ensemble members are tabulated and used to predict the full runtime cost of a complete ensemble member simulation.

In this experiment set, we limit EC2 instance types to m1.small and c1.medium, and run the EnKF with 2 stages, 128 ensemble members. For baseline experiments/performance numbers, we do not try to optimize using adaptivity (neither application-level nor infrastructure adaptivity. Figure 2 depicts the TTC for different values of imposed deadlines and thus gives quantitative information on acceleration provided by the autonomic scheduler to meet deadlines. The



(a) Time To Completion (b) Task consumption (c) EC2 cost

Fig. 2: Results from baseline experiments (without adaptivity) but with a specified deadline. We run the EnKF with 2 stages , 128 ensemble members and limit EC2 instance types to m1.small and c1.medium. Tasks are completed within a given deadline. The shorter the deadline, the more EC2 nodes are allocated.

imposed deadline is assumed to be 120, 108, 96, 72 and 24 minutes respectively; this corresponds to 0%, 10%, 20%, 40% and 80% acceleration, based upon the assumption that a deadline of 120 minutes (0% acceleration) corresponds to a TTC when only TG nodes are used. Figure 2 (a) shows TTC for stage 1 and stage 2 where each stage is heterogeneous and all tasks are completed by the deadline (for all cases). Figure 2 (b) shows the number of tasks completed by the TG and number off-loaded onto EC2, as well as the number of allocated EC2 nodes for each stage. As the deadline becomes shorter, more EC2 nodes are allocated, hence, the number of tasks consumed by EC2 increases. Because m1.small have relatively poorer performance compared to c1.medium, the autonomic scheduler only allocates c1.medium instances for both stages. Figure 2 (c) shows costs incurred on EC2. As more EC2 nodes are used for shorter deadlines, the EC2 cost incurred increases. The results also show that most tasks are off-loaded onto EC2 in order to meet the "tight" deadline of finishing within 24 minutes.

B. Track 1: Infrastructure adaptations

The goal of experiment involving infrastructure-level adaptivity (track-1) is to investigate advantages – performance or otherwise, that may arise from the ability to dynamically select appropriate infrastructure, and possibly vary them between the heterogeneous stages (of the application workflow). Specifically, we see if any additional acceleration (compared to the baseline experiments) can be obtained. In order to provide greater variety in resource selection, we include all EC2 instance types from Table I; these can be selected in stage 2 of the EnKF workflow.

C. Track 2: Adaptations in application execution

In this experiment we run ensemble member rank 0 with variations in solvers/preconditioners and infrastructure. In each case, ensemble rank 0 was run with a solver (generalized minimal residual method GMRES, conjugate gradient CG or biconjugate gradient BiCG), a preconditioner (block Jacobi or no preconditioner) for a given problem size with a varying number of cores (1 through 8). Two infrastructure solutions were available: a single c1.xlarge or four c1.medium instances.

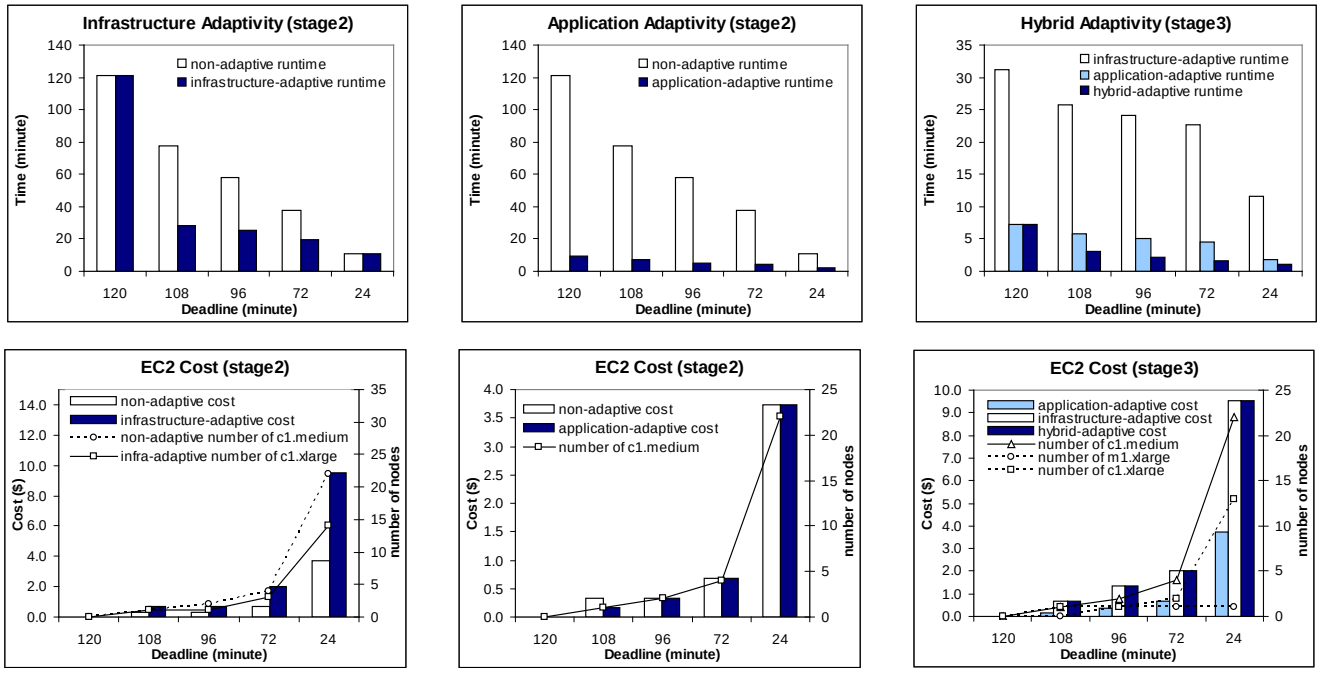
D. Track 3: Adaptations at the application and infrastructure levels

In this experiment set, we explore both infrastructure as well as application-level adaptivity. We try infrastructure-level adaptivity in the first stage, followed by application-level adaptivity in the second stage and hybrid adaptivity in the third stage. In principle the final performance should not be sensitive to the ordering. We will compare hybrid-adaptivity to application-adaptivity, as well as infrastructure-adaptivity.

V. Analysis

A. Track 1: Infrastructure adaptations

Figure 3 (a) shows TTC and EC2 cost, with and without infrastructure adaptivity. Overall, TTC decreases when more resource types are available and infrastructure adaptivity is applied; this can be understood by the fact that the autonomic scheduler can now select more appropriate resource types to utilize. There is no decrease in the TTC when using infrastructure adaptivity for a deadline of 120 because all tasks are still completed by TG node (since it represent 0% acceleration). The difference between the TTC with and without infrastructure-level adaptivity, decrease as the deadline becomes tighter; results show almost no savings with the 24 minutes deadline. This is mostly due to the fact that with a 24 minute deadline, a large number of nodes are allocated thus no further gain can be obtained through infrastructure adaptivity.



(a) Infrastructure adaptivity (b) Application adaptivity (c) Infrastructure/Application adaptivity

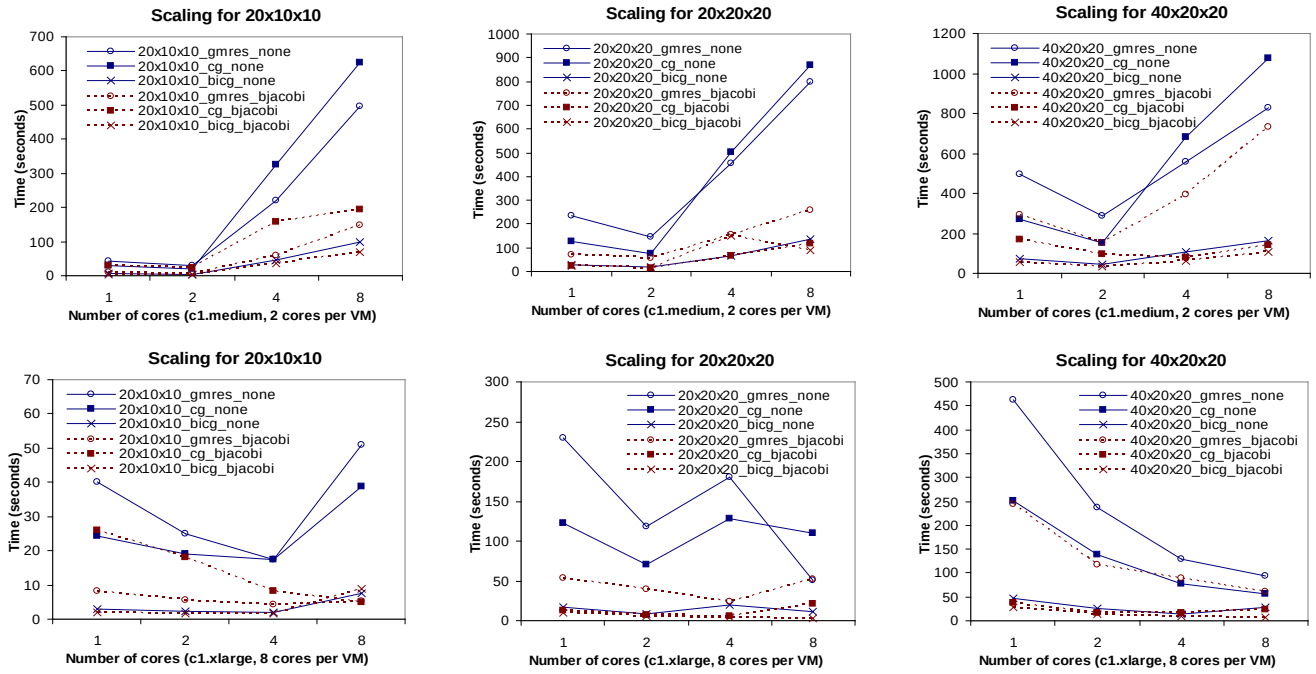
Fig. 3: Experiments with adaptivity. (a) Track 1-Infrastructure adaptivity: we limit EC2 instance types to m1.small and c1.medium for non-adaptive run and use all types described in Table I for infrastructure-adaptive run. The TTC is reduced with infrastructure adaptivity at additional cost. (b) Track 2-Application adaptivity: the optimized application option is used for application-adaptive run. The TTC is reduced with application adaptivity for equivalent or slightly less cost. (c) Track 3- Infrastructure/Application adaptivity: the TTC is reduced further than with application or infrastructure adaptivity on its own. The cost is similar to that in infrastructure adaptivity for durations less than one hour since EC2 usage is billed hourly with a one hour minimum.

Figure 3 (a) also shows the number of nodes allocated for each run and the cost of their usage. The autonomic scheduler selects only c1.xlarge to use for infrastructure-adaptive runs because the runtime estimator predicts that all tasks will run fastest on c1.xlarge. On the other hand the scheduler selects only c1.medium for non-adaptive runs. Infrastructure-adaptive runs cost more than non-adaptive runs, roughly 2.5 times more at the 24 minute deadline even though the number of nodes for non-adaptive runs is larger than the number of nodes for infrastructure-adaptive runs. This is because the hourly cost of c1.xlarge is much higher than c1.medium (see Table I) and both TTCs are rather short (less than half hour). Since we used deadline-based policy, the autonomic scheduler selects the best performing resource regardless of cost; however, when we switch policies to include economic factors in the autonomic scheduler decision making (i.e., considering TTC as well as cost for 24 minutes deadline), the scheduler selects c1.medium instead of c1.xlarge.

B. Track 2: Adaptations in application execution

Figure 4 shows the time to completion of an individual ensemble member, in this case ensemble rank 0 with various solver/preconditioner combinations. For experiments in track-2, we varied over two different types of infrastructure, each with 4 different core counts $(1, 2, 4$ and $8)$ and three problem sizes. We investigated six combinations of solvers and preconditioners over the range of infrastructure. Naturally, there is no one, single, solver/preconditioner combination that works for all problem sizes, infrastructures and core counts. Note the experiments were carried out on different infrastructure, but the infrastructure was not *adaptively* varied. For example, a problem of size 40x20x20 is best solved on 4 cores in a c1.xlarge instance with a BiCG solver and a block Jacobi preconditioner. This is different from the 2 core, BiCG solver and no preconditioner combination for a $20\times10\times10$ problem on a c1.medium VM.

Basic profiling of the application suggests that most of the time is spent in the solver routines, which are communication intensive. As there is no dedicated, high bandwidth, low latency interconnect across instances, MPI performance will suffer, and subsequently MPI intensive



(a) Problem size 20×10×10 (b) Problem size 20×20×20 (c) Problem size 40×20×20

Fig. 4: Time to completion for simulations of various sizes with different solvers (GMRES, CG, BiCG) and block-Jacobi preconditioner. Benchmarks ran on EC2 nodes with MPI. Problem size increases going from left to right. The top row is for a 2 core VM and the bottom row is for an 8 core VM.

solvers. The collective operations in MPI are hit hardest, affecting Gram-Schmidt orthogonalization routines adversely. Conjugate gradient solvers on the other hand are optimal for banded diagonally dominant systems (as is the case in this particular problem) and require less internal iterations to reach convergence tolerance. Figure 4 shows that the performance profile changes as the problem size increases (moving from left to right). Comparison of the profiling data suggests that a smaller percentage of simulation time is spent in communication as the problem size increases. This is obviously due to the fact that there are larger domain partitions for each instance to work on. Detailed profiling of inter–instance MPI bandwidth and latency is still underway however early results suggest that this trend continues.

Figure 3 (b) shows TTC (upper figure) and EC2 cost (lower figure) using application adaptivity. Applying an optimized application configuration reduces TTC considerably. Application configuration does not affect the selection of infrastructure and the number of nodes, hence, the cost depends on TTC. However, since EC2 costs are billed on an hourly basis (with a minimum of one hour), the decrease in cost does not match the decrease in TTC. For example, at 108 minute deadline, the time difference between non-adaptive mode and application-adaptive mode is more than one hour, hence, the cost of application-adaptive mode reduces half. However, because TTCs are all within one hour for other deadlines, EC2 costs are the same for them.

C. Track 3: Adaptations at the application and infrastructure levels

Figure 3 (c) shows TTCs and EC2 costs when both application and infrastructure adaptivities are applied; the white columns correspond to infrastructure adaptivity TTC, the light blue columns correspond to application adaptivity TTC and the dark blue columns correspond to hybrid adaptivity TTC. As mentioned earlier, the application spends most of its time in the iterative solver routine. The application also runs twenty time-steps for each of the 128 simulations. Therefore, the potential for improvement in TTC from solver/preconditioner selection is substantial. As we expect, the TTC of infrastructure-adaptive runs are larger than those of application-adaptive runs, especially since infrastructure adaptivity occurs once per every stage and application adaptivity influences every solver iteration. Both infrastructure and application adaptivity result in TTC reduction, and even more so when used simultaneously. The cost for using infrastructure adaptivity is higher than that of using application adaptivity. This is due to the simple fact that application adaptivity improves the efficiency of the application without the need for an increase in resources. It is worth mentioning that ours is a special case as the application depends heavily on a sparse matrix solve with an iterative solver. Other applications that use explicit methods cannot make use of application adaptivity (no solvers).

VI. Conclusion and Future Work

In summary, by developing and analyzing a dynamic workflow application to by explore a rich set of infrastructure as well as application configurations, we established the benefits of hybrid HPC grids-clouds execution modes. We also investigated application or/and infrastructure adaptivity and determined how the adaptations affect performance as measured by time-to-completion, as well as cost. To achieve these objectives, we built an autonomic adaptation engine inside of CometCloud, consisting of an autonomic adaptivity manager along with the workflow manager, runtime estimator, autonomic scheduler, and grids/clouds agent.

Experimental results show that time-to-completion decreases with both system or application-level adaptivity. We also observe that the time-to-complete decreases further when applying both the system and application-level adaptivity. Furthermore, while EC2 cost decreases when application adaptively is applied, it increases when infrastructure adaptivity is applied. This is despite a reduced time-to-completion, and is because adaptation causes the application to use more expensive instance types.

The new autonomic capabilities added to the CometCloud engine enable it to schedule complex workflows with varying costs, objective functions and satisfy goals such as reduced time-to-completion or reduced total-cost etc. in a hybrid HPC Grid-Cloud environment. While our work so far has been focused on EnKF inverse problem workflow (a fairly straightforward, linear workflow), workflows such as parameter or model surveys can similarly be scheduled. Investigation of more complex workflows and their scheduling is still in the planning phase. Some challenges that we anticipate include scheduling nonlinear and conditional tasks as well as "while–loop" workflows. These types of complex workflows cannot be easily analyzed for cost, and are unpredictable a priori. However, the autonomic and adaptive capabilities we developed will play a major part in resolving these issues.

In the near future we will be working on expanding our cloud infrastructure integration to include Eucalyptus and Nimbus clouds. Proper performance investigation of the reservoir simulator (including MPI message frequency, size, latency, bandwidth and so on) is also being studied. In terms of optimization, work will begin on investigating the effects of network performance for file transfer as well as VM optimization. Finally, we intend to revisit other autonomic objectives (conservation and resilience) with the newly developed autonomic layer.

Acknowledgements

The research presented in this paper is supported in part by National Science Foundation via grants numbers IIP 0758566, CCF-0833039, DMS-0835436, CNS 0426354, IIS 0430826, and CNS 0723594, by Department of Energy via grant numbers DE-FG02-06ER54857 DE-FG02- 04ER46136 (UCoMS), by a grant from UT Battelle, and by an IBM Faculty Award, and was conducted as part of the NSF Center for Autonomic Computing at Rutgers University. Experiments on the Amazon Elastic Compute Cloud (EC2) were supported by a grant from Amazon Web Services and CCT CyberInfrastructure Group grants. SJ and MP would like to acknowledge the e-Science Institute, Edinburgh for supporting the Research theme on Distributed Programming Abstractions. YE would like to acknowledge Dr. Christopher White, Dr. Kent Milfeld and Mr Bob Garza.

References

[1] C. Vecchiola, S. Pandey, and R. Buyya, "High-performance cloud computing: A view of scientific applications," in *Pervasive Systems, Algorithms, and Networks (ISPAN), 2009 10th International Symposium on*, dec. 2009, pp. 4 –16.

[2] S. Ostermann, R. Prodan, and T. Fahringer, "Extending grids with cloud resource management for scientific computing," in *Grid Computing, 2009 10th IEEE/ACM International Conference on*, oct. 2009, pp. 42 –49.

[3] K. Chine, "Scientific computing environments in the age of virtualization toward a universal platform for the cloud," in *Open-source Software for Scientific Computation (OSSC), 2009 IEEE International Workshop on*, sept. 2009, pp. 44 –48.

[4] Cyclone. http://www.sgi.com/products/hpc_cloud/cyclone/.

[5] H. Kim, Y. el Khamra, S. Jha, and M. Parashar, "An autonomic approach to integrated hpc grid and cloud usage," in *e-Science, 2009. e-Science '09. Fifth IEEE International Conference on*, Dec. 2009, pp. 366–373.

[6] Y. E. Khamra and S. Jha, "Title: Developing Autonomic Distributed Scientific Applications: A Case Study From History Matching Using Ensemble Kalman-Filters," in *Sixth International Conference on Autonomic Computing, 2009. ICAC '09 (Barcelona)*. IEEE, 2009.

[7] Y. Y. El-Khamra, "Real-time reservoir characterization and beyond: Cyberinfrastructure tools and technologies," Master's thesis, Louisiana State University, Baton Rouge, Louisiana, 2009.

[8] M. D. de Assuncao, A. di Costanzo, and R. Buyya, "Evaluating the cost-benefit of using cloud computing to extend the capacity of clusters," in *HPDC '09: Proceedings of the 18th ACM international symposium on High performance distributed computing*. New York, NY, USA: ACM, 2009, pp. 141–150.

[9] C. Vazquez, E. Huedo, R. Montero, and I. Llorente, "Dynamic provision of computing resources from grid infrastructures and cloud providers," in *Grid and Pervasive Computing Conference, 2009. GPC '09. Workshops at the*, may 2009, pp. 113 –120.

[10] T. Freeman and K. Keahey, "Flying low: Simple leases with workspace pilot," in *Euro-Par*, 2008, pp. 499–509.

[11] H. Kim, S. Chaudhari, M. Parashar, and C. Marty, "Online risk analytics on the cloud," in *Cluster Computing and the Grid, 2009. CCGRID '09. 9th IEEE/ACM International Symposium on*, May 2009, pp. 484–489.

[12] A. Ozer and C. Ozturan, "An auction based mathematical model and heuristics for resource co-allocation problem in grids and clouds," in *Soft Computing, Computing with Words and Perceptions in System Analysis, Decision and Control, 2009. ICSCCW 2009. Fifth International Conference on*, sept. 2009, pp. 1 –4.

[13] P. Martinaitis, C. Patten, and A. Wendelborn, "Remote interaction and scheduling aspects of cloud based streams," in *E-Science Workshops, 2009 5th IEEE International Conference on*, dec. 2009, pp. 39 –47.

[14] L. Nie and Z. Xu, "An adaptive scheduling mechanism for elastic grid computing," *Semantics, Knowledge and Grid, International Conference on*, vol. 0, pp. 184–191, 2009.

[15] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman, "Grid information services for distributed resource sharing," in *High Performance Distributed Computing, 2001. Proceedings. 10th IEEE International Symposium on*, 2001, pp. 181 –194.

[16] M. Siddiqui, A. Villazon, J. Hofer, and T. Fahringer, "Glare: A grid activity registration, deployment and provisioning framework," *SC Conference*, vol. 0, p. 52, 2005.

[17] T. Dornemann, E. Juhnke, and B. Freisleben, "On-demand resource provisioning for bpel workflows using amazon's elastic compute cloud," in *CCGRID '09: Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 140–147.

[18] S Jha, D S Katz, A Luckow, A Merzky and K Stamou, Understanding Scientific Applications for Cloud Environments, submitted to book on Cloud Computing, Edited by Raj Kumar Buyya, to be published by Wiley
draft available at: http://cct.lsu.edu/~sjha/select_publications/cloud_book_chapter.pdf.

[19] S. Jha, M. Parashar, and O. Rana, "Investigating autonomic behaviours in grid-basedcomputational science applications," in *GMAC '09: Proceedings of the 6th international conference industry session on Grids meets autonomic computing*. New York, NY, USA: ACM, 2009, pp. 29–38.

[20] R. J. Al-Ali, K. Amin, G. von Laszewski, O. F. Rana, D. W. Walker, M. Hategan, and N. J. Zaluzec, "Analysis and Provision of QoS for Distributed Grid Applications," *Journal of Grid Computing (Springer)*, vol. 2, no. 2, pp. 163–182, 2004.

[21] R. E. Kalman, "A new approach to linear filtering and prediction problems." [Online]. Available: http://www.cs.unc.edu/~welch/kalman/media/pdf/Kalman1960.pdf

[22] Y. Gu and D. S. Oliver, "An iterative ensemble kalman filter for multiphase fluid flow data assimilation," *SPE Journal*, vol. 12, no. 4, pp. 438–446, 2007.

[23] X. Li, C. White, Z. Lei, and G. Allen, "Reservoir model updating by ensemble kalman filter-practical approaches using grid computing technology," in *Petroleum Geostatistics 2007*, Cascais,Portugal, August 2007.

[24] Y. Gu and D. S. Oliver, "The ensemble kalman filter for continuous updating of reservoir simulation models," *Journal of Engineering Resources Technology*, vol. 128, no. 1, pp. 79–87, 2006.

[25] H. Kim, M. Parashar, L. Yang, and D. Foran, "Investigating the use of cloudbursts for high throughput medical image registration," in *Proceedings of the 10th IEEE/ACM International Conference on Grid Computing (Grid 2009)*, 2009.

[26] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup protocol for internet applications," in *ACM SIGCOMM*, 2001, pp. 149–160.

[27] C. Schmidt and M. Parashar, "Squid: Enabling search in dht-based systems," *J. Parallel Distrib. Comput.*, vol. 68, no. 7, pp. 962–975, 2008.

[28] N. Carriero and D. Gelernter, "Linda in context," *Commun. ACM*, vol. 32, no. 4, pp. 444–458, 1989.

[29] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[30] Hadoop. http://hadoop.apache.org/core/.

[31] Thain D, Tannenbaum T and Livny M 2005 Distributed Computing in Practice: The Condor Experience Concurrency - Practice and Experience 17 2-4 323-56.

[32] S. Balay, K. Buschelman, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang, "PETSc Web page," 2001, http://www.mcs.anl.gov/petsc.