A2L2: an Application Aware Flexible HPC Scheduling Model for Low-Latency Allocation

Gonzalo P. Rodrigo, Per-Olov Östberg,
Erik Elmroth
Dept. Computing Science, Umeå University
SE-901 87, Umeå, Sweden
{gonzalo, p-o, elmroth}@cs.umu.se

Lavanya Ramakrishnan
Lawrence Berkeley National Lab
Berkeley, CA 94720, USA
lramakrishnan@lbl.gov

ABSTRACT

High-performance computing (HPC) is focused on providing large-scale compute capacity to scientific applications. HPC schedulers tend to be optimized for large parallel batch jobs and, as such, often overlook the requirements of other scientific applications. In this work, we propose a cloud-inspired HPC scheduling model that aims to capture application performance and requirement models (Application Aware - A2) and dynamically resize malleable application resource allocations to be able to support applications with critical performance or deadline requirements. (Low Latency allocation - L2). The proposed model incorporates measures to improve data-intensive applications performance on HPC systems and is derived from a set of cloud scheduling techniques that are identified as applicable in HPC environments. The model places special focus on dynamically malleable applications; data-intensive applications that support dynamic resource allocation without incurring severe performance penalties; which are proposed for fine-grained backfilling and dynamic resource allocation control without job preemption.

Categories and Subject Descriptors

D4.1 [Operating Systems]: Process Management—*Scheduling*

Keywords

Scheduling; job; HPC; malleable; applications; low-latency

1. INTRODUCTION

High-performance computing (HPC) and cloud computing are paradigms focused on large-scale resource provisioning through aggregation of resources in data centers. Although similar at high level, the paradigms have fundamental differences in system objectives and target applications that affect the design of their infrastructure and schedulers.

HPC systems serve scientific applications, traditionally composed of tightly coupled parallel jobs and use batch schedulers focused on system utilization [5]. Cloud computing offers variable, on-demand compute capacity to run different types of applications that often face great variability in load and require resource and infrastructure elasticity [4]. As the data-intensive applications that are common in cloud workloads [31] are becoming increasingly common in HPC [40], a trend towards increased overlap between the paradigms can be observed.

In this work, we focus on scientific applications in HPC systems and, among these, in particular on data-intensive applications. As this application type is characterized by minimal communication between tasks and I/O centric performance models [41], application input data can often be rearranged among the processing nodes without affecting application result or performance (as in, e.g., MapReduce applications [9]). This property allows us to dynamically change application resource allocations without incurring significant performance penalties (lost work or low resource utilization). We call these applications *dynamically malleable* and observe that they provide schedulers interesting opportunities for increased scheduling performance: application malleability provides schedulers the ability to make small adjustments to the size of jobs, which allows for tighter packing of job in backfilling (improved resource utilization). In addition, the ability to downsize jobs allows schedulers to temporarily free (some of the) resources allocated to dynamically malleable applications, which can be used to make room to schedule time-critical jobs with short response times. This enables allocation of resources to run event-synchronized computations (e.g., real-time processing of data from external experiments) without advance reservations of resources (known to reduce resource utilization [1]).

However, this approach also offer challenges: in order to efficiently alter application resource assignments, schedulers require application-level control to adapt to resource changes [34]. Also, to allow schedulers to decide on resource allocations aimed at a specific performance target, users should provide performance expectations rather than resource estimations (e.g., execution deadlines instead of job runtime estimations [12]). In addition, data-intensive applications can expect high performance improvements from data locality (data on compute nodes) [34]. This is hard to realize on HPC systems that typically provide high-performance shared (distributed) file systems rather than local node storage [26]. Finally, today's batch schedulers do not capture the performance and requirement models of data-intensive

Publication rights licensed to ACM. ACM acknowledges that this contribution was authored or co-authored by an employee, contractor or affiliate of the US Government. As such, the Government retains a nonexclusive, royalty-free right to publish or reproduce this article, or to allow others to do so, for Government purposes only. VTDC'15, June 15 - 16, 2015, Portland, OR, USA. Copyright is held by the owner/author(s). Publication rights licensed to ACM. ACM 978-1-4503-3573-7/15/06...\$15.00 DOI: <http://dx.doi.org/10.1145/2755979.2755983>.

	HPC	Cloud
Applications	Batch, data-intensive, parallel tightly coupled jobs.	Batch jobs, data-intensive applications, services.
Platform	Bare metal.	Bare metal, virtualization, execution frameworks.
Hardware	Homogeneous (not in new systems [8]), low-latency networks, Burst Buffer or I/O nodes.	Heterogeneous, commodity, large RAM, storage on node.
Target	High Utilization, low wait times, low cost.	Availability, no wait for services, low cost.
Technique	Job ordering, FCFS, backfilling, monolithic, static partition.	Placement, feedback, elasticity, migration, monolithic, static partition, two-level, shared state.

Table 1: Applications, objectives, and scheduler techniques in HPC and cloud environments.

applications, often resulting in sub-optimal decisions in scheduling of data-intensive applications on HPC systems.

In this paper we present *A2L2*, an *Application Aware flexible HPC scheduling model for Low Latency allocation* of resources for jobs that adapts cloud techniques to tackle HPC scheduling challenges. The model is considered *application aware* as it is built on a two-level cloud inspired model [33] in which each type of application has an individual scheduler that captures its performance and requirement models. The model is considered to be *flexible* as it uses dynamically malleable applications to backfill resource gaps (later described as *flexible backfill* technique). Finally, the model aims to support *low-latency allocations* by using resource expropriation to free resources for applications with time-sensitive or real-time needs. The main contributions of this paper are:

- A2L2, an HPC application-aware scheduling model that makes use of dynamically malleable applications to enable flexible backfilling, low-latency scheduling of jobs, and performance-oriented scheduling of data-intensive applications.
- A discussion of dynamic management and control of malleable applications in HPC environments.
- A comparative discussion of scheduling techniques applicable to both HPC and cloud computing.

The remainder of this paper is structured as follows. Section 2 presents the potential and challenges of managing dynamically malleable applications. Section 3 provides a comparative overview of cloud and HPC resource provisioning (focused on scheduling and placement mechanisms) and identifies a set of key infrastructure and application characteristics that impact design of such mechanisms. Section 4 presents A2L2, a proposed scheduling model aimed to improve the flexibility and efficiency of HPC environments by use of cloud-style placement techniques. Section 5 provides a summary of identified challenges that must be overcome to realize this model and proposes a way forward in this work. Section 6 outlines a brief summary of related work and Section 7 concludes the paper.

2. DYNAMICALLY MALLEABLE

Data-intensive applications are typically organized as workflows and characterized by significant I/O operations. They benefit from parallelism but are not necessarily tightly coupled. In this work, we focus on what we call *dynamically malleable applications*, where input data can be divided in a minimum unit (quantum) and output data only depends on a specific operation or set of operations on that input

data. The processing of each quantum is independent and, as a consequence, processing n quanta will require the same amount of compute time (under the same resource circumstances), no matter whether the computations are done in parallel or serially [41]. Additionally, as each quantum is independent, runtime changes in the job geometry are possible without losing intermediate results (dynamically malleable). However, the application performance model differs from that of tightly coupled parallel batch jobs and application level knowledge is required to control application resource allocations.

2.1 Performance model

The performance of data-intensive applications is by definition I/O bound. In cloud environments, distributed file systems that divide and place data on compute nodes are commonly used to achieve high throughput for data-intensive applications [34]. As such, there are two possible performance models depending on the relationship between the size of the input data and the compute node:

- Storage-centric: Input data does not fit in node memory and is staged onto the local node storage system. Hadoop [34] is an example of an execution framework built around compute node storage based distributed file systems.
- Memory-centric: The input data of all application stages fit in node memory and no local node storage is needed. Spark [42] is an example of an execution framework specialized for this kind of applications that creates a memory-based distributed file system.

As I/O operations on memory are faster, memory-centric applications offer an overall better data processing throughput than storage-centric applications. However, the input dataset size is limited by the total available node memory, memory-centric applications may require more compute nodes to process the same input dataset. If not enough nodes are available, memory-centric applications have to load more input data on the main memory of the compute nodes, reducing system overall performance [42].

However, the key performance enhancement mechanism of data intensive clouds environments (distributed storage) cannot natively be used in HPC as compute nodes typically do not have local storage and large fast shared file systems may not offer optimal performance for an application’s data access patterns [26]. In Section 4.3.2 we present how A2L2 attempts to bring cloud’s performance improvement techniques to HPC.

2.2 Flexibility and management

The degree of parallelism can be changed in dynamically malleable applications, impacting the overall runtime accordingly. As we present in the next sections, this flexibility can be used to meet scheduler objectives. However, this flexibility requires applications to be aware of changes in resources state and availability, and claim and release resources as needed during runtime. In data intensive cloud frameworks (e.g. Hadoop [34]), each job has a master task controlling the rest. The frameworks communicate with the master tasks to coordinate changes in resource allocation.

The scientific community has developed similar tools for certain dynamically malleable applications in HPC, but they are not integrated with resource management system schedulers and, as a consequence, do not support graceful re-scaling of applications during runtime.

3. HPC AND CLOUD SYSTEMS

This work proposes a new scheduling model that focuses on application-aware scheduling to enable low-latency (on-demand) scheduling of applications in HPC environments. The first step is to understand what cloud techniques enable these features, and compare these to the regular HPC scheduling. Table 1 summarizes the observed differences and similarities between HPC and cloud scheduling.

3.1 Batch centric HPC scheduling

HPC systems focus on computing needs of scientific applications. The traditional HPC application is often described as a large simulation that run tightly coupled parallel jobs [35], allocates large numbers of nodes for long periods of time, and requires low-latency synchronized inter-process communication across nodes. Lately however, data analysis applications have become more present on HPC systems. These applications do not have strong coupling requirements, are easier to parallelize, and may in many cases be dynamically malleable (the degree of parallelism can be adapted at runtime) [9].

HPC system architectures are designed to support the scientific applications characteristics (e.g., provide a high enough degree of parallelism to simulate a certain model) and to offer large amounts of compute capacity to large user bases while keeping the cost per compute time-unit low. The resulting systems are typically very efficient in terms of operational, energy, and procurement costs, but the support for diverse applications models is often limited. Typically HPC infrastructures are relatively homogeneous (albeit less so in newer systems [8]), have no storage on compute nodes, and use synchronized networks and high performance parallel storage systems.

HPC schedulers use a combination of techniques aimed to satisfy the system’s objectives. First Come First Serve (FCFS) algorithms execute jobs in arrival order until no more jobs are waiting, or there are not enough available resources to run the first job in the queue [15]. FCFS is often complemented with Backfilling [27], a scheduling technique aimed to increase utilization by searching in the waiting queue for jobs whose resource requirements and estimated runtimes can be met using available resources without delaying the start time of jobs found earlier in the waiting queue. Job prioritization is achieved by adding techniques such as fairshare [29] or priority queues [5]. Also, jobs can

Figure 1: Scheduling of five batch jobs with FCFS and backfilling. Job J5 can be backfilled while J4 can not (as it would delay J3, which arrived before).

incorporate checkpointing techniques that can be used to restart jobs with reduced lost work after preemption.

Figure 1 illustrates scheduling of jobs using FCFS with backfilling. Each box represents a job (y-axis represents the fraction of the total nodes required by the job; x-axis represents the estimated runtime). By following FCFS, J3 starts after J2, leaving a significant amount of resources free to run J4 or J5 when they arrive. The backfilling algorithm finds that, according to its estimated runtime, J4 would end after J3’s estimated start time, while J5 would end before. As a consequence J5 is backfilled. Classic batch schedulers take into account the requested resources (width) and estimated runtime (length) of jobs in a static way. This limits backfilling as it can only be performed if a match with a corresponding geometry can be fit with a resource gap. AL2L aims to address this limitation by adapting dynamically malleable jobs to the available resource windows.

3.2 Application-aware cloud placement

Cloud computing emerged as a way to sell spare compute capacity from under-utilized infrastructures [4] and has since evolved into a model that offers variable, on-demand, accountable, and instantly available resources to run different types of applications using different delivery service models [23], e.g. Infrastructure as a service (IaaS), platform as a service (PaaS) and software as a service (SaaS). Services, batch jobs, and data-intensive applications are some of the types of applications that are common in cloud data centers [31]. Typically, parallel cloud applications are not as tightly coupled as in HPC environments [38] and very variable levels of utilization of the allocated resources are observed [13].

The NIST definition of cloud computing captures its objectives: no wait time to provide resources to services, dynamically increase/decrease resources allocated to an application, and support for diverse applications and platforms at the lowest cost possible. To comply with this definition cloud computing brings a series of required infrastructure characteristics to the data center: commodity hardware, nodes with large RAM capacity (to support multiple VMs on the same node) and high-speed networks (but not low-latency, like in HPC systems). Cloud applications also present various scheduling challenges. Some applications must be run immediately or at deadlines (e.g., services), allocation on-demand imposes the use of prediction methods to perform resource planning [4] and the performance of an application is hard to predict as it depends on the code itself, the supported load and the presence of other applications hosted on

the same resources [43]. However, cloud environments typically also offer opportunities and tools not present in HPC environments. For example, applications can be migrated between compute nodes [4] and applications not fully using their allocated resource capacity can sometimes be over-booked without noticeable performance degradation [36].

Scheduling in cloud datacenter starts as a placement decision. When more capacity is needed by an application it can be allocated a larger share of its host machine (vertical elasticity) or increases the number of instances of the application on other machines (horizontal elasticity). Placement, to optimize a set of target functions (e.g., energy efficiency, resource utilization, quality of service) while satisfying a set of constraints (e.g., memory or affinity requirements of a VM) by assigning resources to applications, is an NP hard problem [24].

Compared to HPC environments, the variability of cloud applications can be seen to bring different performance models and requirements to the cloud data center; e.g., batch jobs can wait to be executed, while services can not; batch job performance is measured by execution time while services focus on quality of service aspects such as response time. There are different approaches to capture applications models in a scheduler [33], with A2L2, we borrow a construct from cloud environments: two level scheduling, i.e. there is one scheduler per application type that interact with a single a resource manger which governs how the resources are assigned to each scheduler. In Section 4.1 we present the model and its underlying challenges.

3.3 HPC trends

Currently, the HPC community is striving towards Exa-scale, i.e. systems with Exaflop peak compute capacity. When reaching this level of capacity the number of cores per node is expected to increase significantly [6] while node memory capacity and I/O latency and bandwidth are expected to not be able grow at the same pace (due to power consumption and technology limitations respectively). Thus, the gap between CPU and I/O capacity is expected to increase, and I/O to become a performance cap for certain applications. As a consequence, a set of solutions for reducing I/O limitations are being investigated for HPC systems:

- Burst buffers on selected compute nodes: solid state memory that behaves like a cache level between main memory and the external IO systems, with larger capacity than the node main memory. The I/O latency of the burst buffer is expected to be significantly smaller than parallel file system's. Applications may decide to use this burst buffer as a local storage system [21].
- I/O dedicated nodes present a similar function to the burst buffers but out of the compute nodes, higher capacity and slightly larger latency [6].

Burst buffers can bring fast storage to (or close to) HPC nodes (i.e. dedicated analysis nodes), which can increase the performance of data-intensive applications as they are often I/O limited. As a side effect, these proposed changes reduce the differences between cloud and HPC environments as some nodes present storage and many cloud infrastructures already incorporate SSDs on their systems [16]. In Section 4.3.2 we explore the idea of using burst buffers in the context of the A2L2 model to increase data-intensive application performance in HPC environments.

Figure 2: A2L2 with three schedulers. The low-latency scheduler requests expropriated resources, which are taken from the dynamically malleable applications.

4. A2L2 SCHEDULING MODEL

Section 3 compares HPC and cloud systems, identifying the methods used in cloud to tackle the challenges induced by the presence of data-intensive applications. In this section we discuss techniques to realize A2L2: a cloud-inspired scheduling model aimed to enable application-aware scheduling of heterogeneous workloads, that takes advantage of the presence of dynamically malleable applications to improve utilization and enable low-latency allocation of resources.

4.1 Application-aware scheduling

The core of the A2L2 approach is a two level resource manager model that serves multiple schedulers at the same time. Each application is managed by a scheduler specific for that application type that captures the application's performance characteristics and requirements. The resource manager controls the resource allocation among schedulers. An example with three schedulers is presented in Figure 2. This schema is a hybrid between the two level and shared state approaches presented in [33] to schedule heterogeneous workloads (services, batch jobs, workflows) on cloud infrastructures. At the core of our approach is the resource manager, which requires the following system characteristics to enable application-aware scheduling and the other features presented in this work:

- More than one scheduler uses the resource manager.
- The resource manager offers a stable and consistent view of the state of the resources to all schedulers.
- All resource allocations follow a request and offer resource protocol jointly controlled by the resource manager and the schedulers. Periodically the resource manager allow all the schedulers to request resources at the same time. This process has two stages. First, *schedulers request* free resources to run an application and, if not all requests can be satisfied, a conflict resolving method is applied (described later). Then, the *resource borrowing* phase starts: schedulers are offered any remaining free resources to increase the allocation of already running applications. Borrowed resources are considered free in the next request phase and it is the responsibility of the allocating scheduler to de-allocate borrowed resources when needed.

- Dynamically malleable application schedulers must support *resource expropriation* - de-allocation of resources at the request of the resource manager.
- Schedulers can request expropriated resources to run time-critical applications if there are not enough free resources to run it.

In our model, each scheduler is isolated from the rest, their decisions are made solely on their individual performance models and policies. The resource manager controls the resources assigned to each scheduler. This control is enforced by individual decision to solve resource request conflicts between schedulers (Conflicts arise when there are not enough resources to satisfy all the resource requests). The aggregated effect of all decisions is intended to produce a resource sharing policy. These are some of the resource sharing policies to be explored in our work:

- Weighted random: Probabilities of each scheduler are configured. Schedulers with higher probability will submit more jobs regardless of their resource time consumption.
- Fairshare: Each scheduler should be able to use a pre-defined share of the resource time for a given time window (e.g. 20% of the produced core-hours in one day). The resource manager keeps track of the past consumed resource time and decisions are aimed to bring the system to the target shares [29]. Schedulers' shares can be adjusted to different values on periods of time to enforce a temporary higher presence of certain types of applications.

This two-level model may produce clear benefits. Although at the core designed for a cloud type usage scenario, high utilization is enforced by the schedulers, that are built around the HPC applications performance models and the objectives of high utilization and capacity. The flexible backfilling may increase the utilization over classic backfilling. At the same time, the model separates how applications share the system from application specific policies (classic batch schedulers aggregate the application specific priorities dissolving both the effect of resource sharing policies and individual application sharing).

In addition, this model supersedes the batch scheduler as it allows application specific scheduling, i.e. introduction of new types of applications do not require a change of the whole scheduler system (merely the addition of a new scheduler for that application type). Also, it provides a level of fault-tolerance as the parts are decoupled and can easily be replicated. It aims to enforce a usage share among applications through the different policies on the resource conflict resolution. Finally, flexible backfilling and resource expropriation has great potential for increasing resource utilization and adding capabilities to the system (more details in the next sections).

4.2 Dynamic allocation of resources

Classic HPC jobs receive resource allocations that remain constant throughout the entire job make-span. In contrast, cloud systems are capable of increasing and reducing the resource allocations of applications to cope with variations in load and demand. Use of dynamically malleable applications in dynamic resource management has the potential for

Figure 3: J1 and J2 borrow free resources (flexible backfilling). J2 returns the borrowed resources when the batch scheduler requests them.

Figure 4: Resources expropriated from two jobs. After the streamed job is completed, the expropriated resources are returned.

disrupting how HPC resources are scheduled and enabling more flexible backfilling and resource expropriation for time-critical applications.

4.2.1 Flexible backfilling

As presented in Section 3.1, resource fragmentation is a problem that can occur to batch schedulers. We propose the use of flexible backfilling: resource allocations for dynamically malleable jobs are increased (with little performance penalty) to use non-allocated resources without delaying other applications. At the end of the resource request phase, the resource manager re-offers unused (but not borrowed) resources to schedulers as *borrowable resources* that can be used to increase the allocations of already running applications. However, as borrowed resources are still considered available (free) for new applications allocations, the resource manager may require schedulers to de-allocate borrowed resources at any time. The theoretical result of this technique is illustrated in Figure 3, where applications with changing resource allocations can be seen in contraposition to the classic batch scheduler example of Figure 1. In this example, two dynamically malleable applications (stacked on the upper part of the figure) reduce their resource allocations, which are used to schedule a *streamed job*. Once the job finishes, one of the applications from which the resources were expropriated is still running and the resources are returned to that application. Remaining resources are given to another dynamically malleable application.

4.2.2 Low-latency allocation

Many scientific experiments produce large amounts of data that are stored for later analysis on HPC systems. However, some experiments would benefit from processing data while

the experiment is running. This can be achieved by aligning the experiment with an advance reservation of HPC resources. However, the use of advance reservations frequently leads to low utilization as they cannot fit the exact experiment time window [1]. Instead, a special type of job submission that aims for jobs to be run within a short time period is needed (equivalent to the on-demand nature of cloud): low-latency allocation of resources.

Low-latency allocation of resources are added to our model by using *resource expropriation* for temporary partial preemption of dynamically malleable jobs resources. A step-by-step example of this technique can be followed in Figure 2 and its corresponding effect observed in Figure 4: the low-latency scheduler requests resources from the resource manager, which expropriates resources from the dynamically malleable applications scheduler. The resource manager evaluates what resource reduction will impact application performance objectives the least and enforce a decision on the two applications. The resources are then used by the low-latency application. Once this application terminates, the expropriated resources are returned to the original scheduler. This scheduler re-assigns these resources to the dynamically malleable applications.

In batch jobs, partial preemption brings the negative effect of lost work. However, the technique is only used on dynamically malleable applications whose performance model allows their resource pool to be resized with a small performance penalty. If a processing node is stopped, its produced output data can be stored and its remaining input data transferred to another node to be processed. The overall performance of the job decreases but no work is lost. The application can continue with reduced resources, but if more resources are allocated to it, its performance may return to the pre-expropriation level.

This technique is inspired by cloud techniques such as Brownout [17] (graceful degradation of application quality of service as a means to cope with increased resource load) and overbooking [36] (packing physical hosts with VMs which do not fully utilize their allocation to better utilize the resources).

4.3 Dynamically malleable management

Dynamically malleable jobs do not have a hard constraint on the required resources. A larger resource allocation implies a shorter running time and vice versa. In classical HPC scenarios, users specify the required resource allocation and estimated runtime for jobs. This poses two challenges. First, users have to estimate the resource allocation and consequent runtime for an application whose performance is hard to predict (and thus, easy to under/overestimate). Second, it eliminates the freedom of the scheduler to adapt the job's allocation to achieve overall targets.

We propose performance-based management for dynamically malleable applications: users provide deadline by when the job should be completed and application schedulers have the freedom to dynamically allocate resources to application so deadlines are met. The benefits for the scheduling model are the following:

- Eliminate user over/under estimations: Workflows run multiple phases with different operations on resources where performance might be unknown. It is complex to estimate the required resources and runtime for each phase. However, if the user provides a deadline, the

scheduler can adapt the resource allocation dynamically and avoid over/under allocation.

- Malleable applications can be used for flexible backfilling or as a source of expropriated resources.

4.3.1 Deadline based resource management

In the proposed A2L2 approach, an specific scheduler will exist for dynamically malleable applications. This scheduler has two goals: to control and manage dynamically malleable jobs (job manager) and calculate the resources needed for these jobs to meet their deadlines (resource calculator). The job manager is meant to be built over one of the existing cloud execution frameworks for dynamically malleable applications (e.g., Hadoop, Spark). It will have the responsibility of controlling the job workers on each allocated node, manage the distribution of input data among them, allocate new resources to a job, and de-allocate them if needed. The scheduling functions will interact with the job manager to free resources or the possibility of using borrowed ones.

The resource calculator will inform the job manager on the resources allocated to the jobs to meet the user provided deadlines. Job deadlines are a well known model for expressing an execution target [12]. Deadlines can be synchronized with real life events and, hence, easy to understand by the user. We propose to study what methodology should be used to perform these calculations and enforce them on dynamically malleable workflows. As presented in previous work [12], it implies translating the overall deadline to per-stage deadlines and estimating the resource performance for each stage. The resources performance estimation is used to calculate the required resource size to meet the deadline of a running stage workflow. Finally, a job manager uses the calculation to alter the resource set assigned to the stage.

Finally, the dynamic malleable scheduler must support resource expropriation and resource borrowing. One of the challenges will be to determine the best heuristic to calculate what resources and from which jobs should be released to minimize the impact in terms of job's execution slowdown. A similar heuristic will be investigated to determine the jobs that should receive extra resources in case the scheduler can borrow resources.

4.3.2 Performance enhancements

The throughput of dynamically malleable applications highly depend on the system's I/O performance. In cloud environments, data locality is often leveraged though distributed file systems [16]. However, in most HPC systems, compute nodes use remote parallel file systems and have little or no local storage on nodes. Application performance will depend on the data access patterns [26] and the network capacity of the data center. This is especially important in Exascale systems as I/O operations will be more expensive at that level [6].

Another performance concern comes from A2L2's ability to dynamically change the resource allocation of applications. In this case, although the performance penalty is smaller than that of full preemption of jobs, the time to recreate the application data on nodes may affect the overall performance.

In order to address these two challenges we propose to use memory-centric approaches when possible. For storage-centric applications, burst buffers and I/O nodes can host a filesystem, equivalent to the distributed file system used in

cloud environments. Application data has to be preloaded on nodes but, once loaded, the application read/write patterns will not impact the performance of the I/O network or the central parallel file system [19]. Also, this distributed file system can be used in a semi-persistent way: if a node is expropriated from an application, the file system is not erased. Once the node is free and the application is restored, the application can keep using the near storage system. The only performance penalty will come from checking what input data has been already processed. This penalty can be neutralized by using eventual data consistency techniques on the distributed file system.

5. CHALLENGES AND FUTURE WORK

Based on the A2L2 model outlined in this paper, further research poses three immediate challenges: further detailed modeling of the different components of the scheduling environment, implementation of the proposed model, and evaluation of the quality of scheduling produced. These challenges are embodied in the various system components for an HPC resource management schema with batch job scheduling, dynamic resource allocation on a per job basis, application level control, resource borrowing, flexible backfilling, resource expropriation, low-latency job allocation, workflow aware scheduling, and data locality enforcement for data-intensive applications on HPC.

The work and research on these challenges cannot be performed as a single line of work, but the aggregation of three parallel lines of work: the first is to characterize the workload of a set of reference systems both on HPC and cloud workloads. We plan to study traces from systems at NERSC (the US National Energy Research Computing Center), the Parallel Workload Archive [10], HPC2N (the High Performance Computing Center North, Sweden), and Google cluster data [39]. This line of investigation has two purposes: performing a comparison between HPC and cloud (to verify the feasibility of using cloud techniques in HPC) and producing a model for synthetic workloads to test our scheduler. Our previous works has initiated this line of research with the workload analysis of two NERSC systems with an special focus on its evolution throughout the systems' lifetime [32].

The second research line aims at building a resource performance model that includes node compute capacity, memory, network, and I/O capabilities. In particular, analysis of the effect on the memory and I/O models versus the presence of node burst buffers, as well as the expected Exascale increases in the gap between capacity of processing units and I/O systems, are topics for study.

The third line aims at creating and evaluating three components: a scheduling suite (implementing our model), a resource emulator (based on the resource model) and a workload emulator (that captures characteristics of workload model). The scheduling suit will be based on community open source scheduling software and will be developed for real system deployment. The resource emulator will wrap the scheduler to run it in a test environment. The workload emulator will model different types of job submissions and user behaviors. This approach will support the implementation test and algorithms evaluation processes. Each model feature will be evaluated comparing its performance (in terms of utilization, turnaround time or enabled capacity) against a reference scenario using a classical HPC scheduling algo-

gorithms. The ultimate goal of this effort to produce a software suite to run over a real HPC system.

6. RELATED WORK

Applications whose resource allocation needs are variable are not new to HPC centers. In 1996 Feitelson and Rudolph [11] did a classification that identifies three types of jobs with different degrees of flexibility: moldable, malleable, and evolving.

Moldable applications [7, 18], are those whose degree of parallelism can be chosen just before they are started but do not support any changes after that [11]. A scheduler can decide on the geometry of a moldable job by considering the current state of the resources, the geometry of waiting jobs, and the system's target functions: e.g., high utilization, or short turnaround time. However, unlike dynamically malleable applications, moldable applications don't allow any changes of their resource allocations once allocated. This disables any further scheduling decisions for them (apart from abortion of the job).

Malleable applications support changes in their resource set during execution time without stopping execution [11]. An example are MPI malleable applications [37, 22]. We define dynamic malleable applications as a subset of of the malleable applications characterized for being data intense and organized in a workflow

These applications requires scheduler support as they need to be aware of any resource offering or reduction. This support has appeared in highly distributed works such as grid scheduling [28, 3] but not in pure HPC schedulers. In this work, we propose a model exclusively for HPC systems, with a fixed size resource set and no possibility to schedule jobs to external resources.

Evolving jobs have resource requirement changes throughout their execution time [11] and the system must satisfy them or their execution will not continue. Some grid scheduling systems offer support to these type of applications [20]. HPC schedulers have limited support for evolving jobs [30] and it is based on advance resource reservations. Our model proposal for HPC offers a possibility of freeing resources from dynamically malleable applications to serve evolving jobs or other rigid jobs with higher priority without using advance reservations.

Finally, multi-level scheduling approaches have been investigated in the past in the grid [2, 25] and cloud communities [14, 33]. The resulting schedulers cannot be directly applied to a HPC system. However, the two-level architecture of A2L2 is inspired by some of the models presented in that work.

7. CONCLUSIONS

This paper presents a comparative analysis between HPC and cloud scheduling methods and extrapolates on insights from this comparison to outline A2L2: a scheduling model for HPC systems that takes application characteristics into consideration when realizing low-latency, on-demand allocation of resources in HPC environments. The first insight of this analysis is that applications with different performance models are present in both HPC and cloud environments. To address this, we propose a multi-level HPC scheduler model that separates application specific policies from how compute time is shared by application types. The second in-

sight is the proposal to use the flexible nature of dynamically malleable jobs to enable different features in HPC scheduling. The presence of jobs whose resource allocations can be scaled up and down during execution, with very low performance penalties, allows both a more flexible backfilling model (where job sizes are changed to free space for backfilling jobs as well as to improve the fit of backfilling jobs to resource allocation gaps) and use of resource expropriation in order to allow admission of time-critical application jobs on-demand.

8. ACKNOWLEDGMENTS

This work is funded by the Swedish Government's strategic effort eSENCE, the Swedish Research Council (VR) under contract number C0590801 for the project Cloud Control, the European Union's Seventh Framework Programme under grant agreement 610711 (CACTOS), and the Office of Science, Office of Advanced Scientific Computing Research (ASCR) of the U.S. Department of Energy under Contract Number DE-AC02-05CH11231.

9. REFERENCES

- [1] M. A. Bauer, A. Biem, S. McIntyre, N. Tamura, and Y. Xie. High-performance parallel and stream processing of x-ray microdiffraction data on multicores. In *Journal of Physics: Conference Series*, volume 341, page 012025. IOP Publishing, 2012.
- [2] F. Berman, R. Wolski, H. Casanova, W. Cirne, H. Dail, M. Faerman, S. Figueira, J. Hayes, G. Obertelli, J. Schopf, et al. Adaptive computing on the grid using AppLeS. *IEEE Transactions on Parallel and Distributed Systems*, 14(4):369–382, 2003.
- [3] J. Buisson, O. Sonmez, H. Mohamed, W. Lammers, and D. Epema. Scheduling malleable applications in multicloud systems. In *2007 IEEE International Conference on Cluster Computing*, pages 372–381. IEEE, 2007.
- [4] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation computer systems*, 25(6):599–616, 2009.
- [5] N. Capit, G. Da Costa, Y. Georgiou, G. Huard, C. Martin, G. Mounié, P. Neyron, and O. Richard. A batch scheduler with high level components. In *IEEE International Symposium on Cluster Computing and the Grid. CCGRID 2005*, volume 2, pages 776–783. IEEE, 2005.
- [6] Y. Chen. Towards scalable I/O architecture for exascale systems. In *Proceedings of the 2011 ACM international workshop on many-task computing on grids and supercomputers*, pages 43–48. ACM, 2011.
- [7] W. Cirne and F. Berman. A model for moldable supercomputer jobs. In *Proceedings 15th International Parallel and Distributed Processing Symposium*. IEEE, 2001.
- [8] J. Dongarra et al. The international exascale software project roadmap. *International Journal of High Performance Computing Applications*, 2011.
- [9] J. Ekanayake, S. Pallickara, and G. Fox. Mapreduce for data intensive scientific analyses. In *IEEE Fourth International Conference on eScience*, pages 277–284. IEEE, 2008.
- [10] D. Feitelson. Parallel workloads archive. 71(86):337–360, 2007. <http://www.cs.huji.ac.il/labs/parallel/workload>.
- [11] D. G. Feitelson and L. Rudolph. Toward convergence in job schedulers for parallel supercomputers. In *Job Scheduling Strategies for Parallel Processing*, pages 1–26. Springer, 1996.
- [12] A. D. Ferguson, P. Bodik, S. Kandula, E. Boutin, and R. Fonseca. Jockey: guaranteed job latency in data parallel clusters. In *Proceedings of the 7th ACM european conference on Computer Systems*, pages 99–112. ACM, 2012.
- [13] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel. The cost of a cloud: research problems in data center networks. *ACM SIGCOMM computer communication review*, 39(1):68–73, 2008.
- [14] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. H. Katz, S. Shenker, and I. Stoica. Mesos: A platform for fine-grained resource sharing in the data center. In *Networked Systems Design and Implementation, (NSDI)*, volume 11, pages 22–22, 2011.
- [15] M. Hovestadt, O. Kao, A. Keller, and A. Streit. Scheduling in HPC resource management systems: Queuing vs. planning. In *Job Scheduling Strategies for Parallel Processing*, pages 1–20. Springer, 2003.
- [16] S.-H. Kang, D.-H. Koo, W.-H. Kang, and S.-W. Lee. A case for flash memory ssd in hadoop applications. *International Journal of Control and Automation*, 6(1):201–210, 2013.
- [17] C. Klein, M. Maggio, K.-E. Årzén, and F. Hernández-Rodríguez. Brownout: Building more robust cloud applications. In *Proceedings of the 36th International Conference on Software Engineering*, pages 700–711. ACM, 2014.
- [18] C. Klein and C. Perez. An RMS architecture for efficiently supporting complex-moldable applications. In *2011 IEEE 13th International Conference on High Performance Computing and Communications (HPCC)*, pages 211–220. IEEE, 2011.
- [19] S. Krishnan, M. Tatineni, and C. Baru. myHadoop-Hadoop-on-Demand on Traditional HPC Resources. *San Diego Supercomputer Center Technical Report TR-2011-2*, University of California, San Diego, 2011.
- [20] W. Lammers. *Adding support for new application types to the Koala grid scheduler*. PhD thesis, Citeseer, 2005.
- [21] N. Liu, J. Cope, P. Carns, C. Carothers, R. Ross, G. Grider, A. Crume, and C. Maltzahn. On the role of burst buffers in leadership-class storage systems. In *IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–11. IEEE, 2012.
- [22] K. Maghraoui, T. J. Desell, B. K. Szymanski, and C. A. Varela. Dynamic malleability in iterative mpi applications. In *7th Int. Symposium on Cluster Computing and the Grid*, pages 591–598, 2007.
- [23] P. Mell and T. Grance. The NIST definition of cloud computing. 2011.

- [24] K. Mills, J. Filliben, and C. Dabrowski. Comparing vm-placement algorithms for on-demand clouds. In *IEEE Third International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 91–98. IEEE, 2011.
- [25] H. Mohamed and D. Epema. Koala: a co-allocating grid scheduler. *Concurrency and Computation: Practice and Experience*, 20(16):1851–1876, 2008.
- [26] E. Molina-Estolano, M. Gokhale, C. Maltzahn, J. May, J. Bent, and S. Brandt. Mixing hadoop and hpc workloads on parallel filesystems. In *Proceedings of the 4th Annual Workshop on Petascale Data Storage*, pages 1–5. ACM, 2009.
- [27] A. Mu’alem and D. Feitelson. Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. *IEEE transactions on parallel and distributed systems*, 12(6):529–543, 2001.
- [28] A. P. Nascimento, A. C. Sena, C. Boeres, and V. E. Rebello. Distributed and dynamic self-scheduling of parallel mpi grid applications. *Concurrency and Computation: Practice and Experience*, 19(14):1955–1974, 2007.
- [29] P.-O. Östberg, D. Espling, and E. Elmroth. Decentralized scalable fairshare scheduling. *Future Generation Computer Systems - The International Journal of Grid Computing and eScience*, 29:130–143, 2013.
- [30] S. Prabhakaran, M. Iqbal, S. Rinke, C. Windisch, and F. Wolf. A batch system with fair scheduling for evolving applications. In *2014 43rd International Conference on Parallel Processing (ICPP)*, pages 351–360. IEEE, 2014.
- [31] B. P. Rimal, E. Choi, and I. Lumb. A taxonomy and survey of cloud computing systems. In *Fifth International Joint Conference on INC, IMS and IDC*, pages 44–51. IEEE, 2009.
- [32] G. Rodrigo, P.-O. Östberg, E. Elmroth, K. Antypass, R. Gerber, and L. Ramakrishnan. HPC system lifetime story: Workload characterization and evolutionary analyses on NERSC systems. In *The 24th International ACM Symposium on High-Performance Distributed Computing (HPDC)*, 2015.
- [33] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes. Omega: flexible, scalable schedulers for large compute clusters. In *Proceedings of the 8th ACM European Conference on Computer Systems*, pages 351–364. ACM, 2013.
- [34] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The hadoop distributed file system. In *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–10. IEEE, 2010.
- [35] A. Snaveley, X. Gao, C. Lee, L. Carrington, N. Wolter, J. Labarta, J. Gimenez, and P. Jones. Performance modeling of HPC applications. *Advances in Parallel Computing*, 13:777–784, 2004.
- [36] L. Tomás and J. Tordsson. Improving cloud infrastructure utilization through overbooking. In *Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference*, page 5. ACM, 2013.
- [37] G. Utrera, J. Corbalan, and J. Labarta. Implementing malleability on MPI jobs. In *Proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques*, pages 215–224. IEEE Computer Society, 2004.
- [38] L. Wang, J. Tao, M. Kunze, A. C. Castellanos, D. Kramer, and W. Karl. Scientific cloud computing: Early definition and experience. In *High Performance Computing and Communications*, volume 8, pages 825–830, 2008.
- [39] J. Wilkes. More Google cluster data. Google research blog, Nov. 2011. Posted at <http://googleresearch.blogspot.com/2011/11/more-google-cluster-data.html>.
- [40] R. Williams, I. Gorton, P. Greenfield, and A. Szalay. Guest editors’ introduction: Data-intensive computing in the 21st century. *Computer*, 41(4):0030–32, 2008.
- [41] J. Wolf, D. Rajan, K. Hildrum, R. Khandekar, V. Kumar, S. Parekh, K.-L. Wu, and A. Balmin. Flex: A slot allocation scheduling optimizer for mapreduce workloads. In *Middleware 2010*, pages 1–20. Springer, 2010.
- [42] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: cluster computing with working sets. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, pages 10–10, 2010.
- [43] X. Zhang, E. Tune, R. Hagmann, R. Jnagal, V. Gokhale, and J. Wilkes. Cpi 2: CPU performance isolation for shared compute clusters. In *Proceedings of the 8th ACM European Conference on Computer Systems*, pages 379–391. ACM, 2013.