Harnessing Grid Resources with Data-Centric Task Farms

Ioan Raicu
Department of Computer Science
University of Chicago
iraicu@cs.uchicago.edu

Committee Members:

Ian Foster: Research Advisor, Department of Computer Science, University of Chicago & Math and Computer Science Division, Argonne National Laboratory, foster@mcs.anl.gov

Rick Stevens: Department of Computer Science, University of Chicago, Math and Computer Science Division, Argonne National Laboratory, stevens@mcs.anl.gov

Alex Szalay: Department of Physics and Astronomy, The Johns Hopkins University, szalay@jhu.edu

Abstract

As the size of scientific data sets and the resources required for analysis increase, data locality becomes crucial to the efficient use of large scale distributed systems for scientific and data-intensive applications. In order to support interactive analysis of large quantities of data in many scientific disciplines, we propose a data diffusion approach, in which the resources required for data analysis are acquired dynamically, in response to demand. Acquired resources (compute and storage) can be "cached" for some time, thus allowing more rapid responses to subsequent requests. We define an abstract model for data-centric task farms as a common parallel pattern that drives the independent computational tasks, taking into consideration the data locality in order to optimize the performance of the analysis of large datasets. This approach can provide the benefits of dedicated hardware without the associated high costs. We will validate our abstract model through discrete-event simulations; we expect simulations to show the model is both efficient and scalable given a wide range of simulation parameters. To explore the practical realization of our abstract model, we have developed a Fast and Light-weight tasK executiON framework (Falkon). Falkon provides for dynamic acquisition and release of resources, data management capabilities, and the dispatch of analysis tasks via a data-aware scheduler. We have integrated Falkon into the Swift parallel programming system in order to leverage a large number of applications from various domains (astronomy, astro-physics, medicine, chemistry, economics, etc) which cover a variety of different datasets, workloads, and analysis codes. We believe our data-centric task farm model to generalize to many domains and applications, and could offer application end-to-end performance improvements, higher resource utilization, improved efficiency, and better application scalability.

1 Introduction

Scientific and data-intensive applications often require exploratory analysis on large datasets. Such analysis is often carried out on large scale distributed resources where data locality is crucial in achieving high system throughput and performance. [9] We propose a "data diffusion" [78, 79] approach that acquires resources for data analysis dynamically, schedules computations as close to data as possible, and replicates data in response to workloads. As demand increases, more resources (computer and associated storage, and data) are acquired and "cached" to allow faster response to subsequent requests. Resources are acquired either "locally" if available, or "remotely" if not; the location only matters in terms of associated cost tradeoffs. Both data and applications can diffuse from low-cost archival or slower disk storage to newly acquired resources for processing. If demand drops, resources can be released, allowing

for their use for other purposes. This approach can provide the benefits of dedicated hardware without the associated high costs, depending crucially on the nature of application workloads and the performance characteristics of the underlying infrastructure.

This data diffusion concept is reminiscent of cooperative Web-caching [16] and peer-to-peer storage systems [14]. Other data-aware scheduling approaches assume static or dedicated resources [1, 2, 3, 4, 10, 5, 70], which can be expensive and inefficient (in terms of resource utilization) if load varies significantly, where our dynamic resource allocation alleviates the problem. The challenges to our approach are that we need to co-allocate storage resources with computation resources in order to enable the efficient analysis of possibly terabytes of data without prior knowledge of the characteristics of application workloads.

To explore the proposed data diffusion, we have developed Falkon [6, 11], which provides dynamic acquisition and release of resources ("executors") and the dispatch of analysis tasks to those executors. We have extended Falkon to allow executors to cache data to local disks, and perform task dispatch via a data-aware scheduler. The integration of Falkon and the Swift parallel programming system [13] provides us with access to a large number of applications from astronomy [7, 8, 12, 13], astro-physics, medicine [13], and other domains, with varying datasets, workloads, and analysis codes.

1.1 Motivations and Challenges

In order to achieve the proposed data diffusion, we have identified three key concepts that must be present in the successful realization of data diffusion. These three concepts are: 1) task dispatch and execution, 2) dynamic resource provisioning, and 3) data caching. We believe that all these are necessary in the practical realization of data diffusion, and hence this sub-section covers the motivation and challenges presented by each.

1.1.1 Task Dispatch & Execution

Many interesting computations can be expressed conveniently as data-driven task graphs, in which individual tasks wait for input to be available, perform computation, and produce output. Systems such as DAGMan [18], Karajan [19], Swift [13], and VDS [[20] support this model. These systems have all been used to encode and execute thousands to hundreds of thousands of individual tasks.

In such task graphs, as well as in the popular master-worker model [21], many tasks may be logically executable at once. Such tasks may be dispatched to a parallel compute cluster or (via the use of grid protocols [22]) to many such clusters. The batch schedulers used to manage such clusters receive individual tasks, dispatch them to idle processors, and notify clients when execution is complete.

This strategy of dispatching tasks directly to batch schedulers has three disadvantages. First, because a typical batch scheduler provides rich functionality (e.g., multiple queues, flexible task dispatch policies, accounting, per-task resource limits), the time required to dispatch a task can be large—30 secs or more—and the aggregate throughput relatively low (perhaps one task/sec). Second, while batch schedulers may support different queues and policies, the policies implemented in a particular instantiation may not be optimized for many tasks. For example, a scheduler may allow only a modest number of concurrent submissions for a single user. Third, the average wait time of grid jobs is higher in practice than the predictions from simulation-based research. [71] These factors can cause problems when dealing with application workloads that contain a large number of tasks.

Full-featured local resource managers (LRMs) such as Condor [18], Condor-J2 [27], PBS [28], LSF [29] support client specification of resource requirements, data staging, process migration, check-pointing, accounting, and daemon fault recovery. These LRMs can maintain throughputs on the order of 0.5~2 tasks/sec, while the latest development version of these same LRMs can do one order of magnitude better with 11~22 tasks/sec.

As part of our work, we propose a streamline task submission framework (Falkon) whose focus is solely the efficient task dispatch. This narrow focus is possible because Falkon can rely on LRMs for certain functions (e.g., accounting) and clients for others (e.g., recovery, data staging). In contrast with LRMs performance, Falkon achieves one to two orders of magnitude higher performance (487 tasks/sec) [6]. With workload specific knowledge (i.e. task length distribution), the throughput can be increased further to over 2500 tasks/sec by amortizing the communication overhead over many tasks. We have also experimented with a light-weight TCP-based communication protocol (as a replacement to Web Services), which we found to further improve throughput to about 5000 tasks/sec. We also have ideas on how to parallelize the architecture, which should allow us to push the overall system throughput well beyond the levels we have achieved so far.

In order to emphasize the need for such high throughputs, we showcase (Figure 1) the achieved resource efficiency at four different scales (small grid sites – 100 processors, medium size grid site – 1K processors, large Grid – 10K processors, and supercomputer – 100K processors) for various throughputs (1, 10, 100, 500, 1K, 10K, 100K, and 1M tasks/sec). It is worth noting that current production LRMs require relatively long tasks in order to maintain high efficiency. For example, even in a small Grid site with 100 processors, tasks need to be 100 seconds in duration just to get 90% efficiency; the task duration is increased to 900 seconds for a modest 1K processors, 10K seconds (~2.8 hours) for 10K processors, and more than 100K seconds (1+ days) for 100K processors just to maintain 90% efficiency. With throughputs in the range of 500 tasks/sec (which is obtainable with our proposed framework – Falkon), the same 90% efficiency can be reached with tasks of length 0.2 seconds, 1.9 seconds, 20 seconds, and 200 seconds for the same four cases.

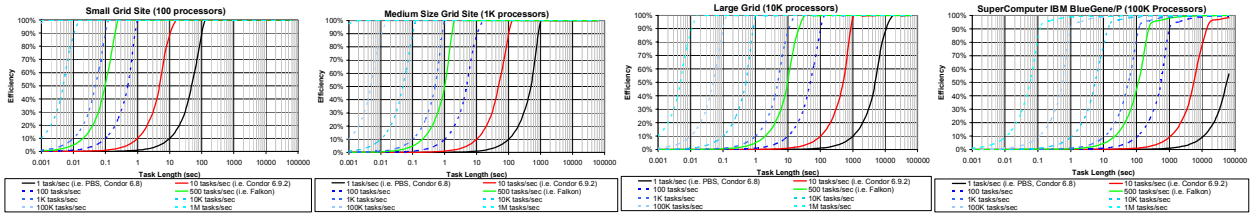


Figure 1: Resource Efficiency for various resource scales and workload characteristics

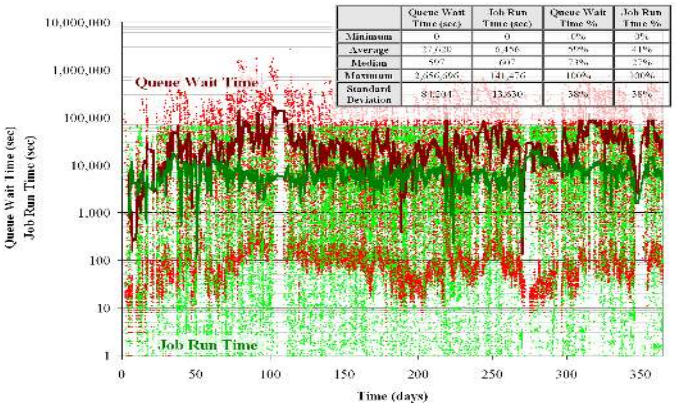
It should be evident that the higher the throughput of tasks/sec that can be dispatched and executed over a set of resources, the higher the resource efficiency for the same workloads and the faster the applications turn-around times will be. Our work with Falkon seeks to reduce task dispatch time by using a streamlined dispatcher that eliminates support for features such as multiple queues, priorities, accounting, etc. Furthermore, we will investigate methods to distribute the dispatch functionality so that dispatch rates could grow further to efficiently manage computational resources in the largest Grids and supercomputers of tomorrow.

1.1.2 Dynamic Resource Provisioning

Batch schedulers commonly used to manage access to parallel computing clusters are not typically configured to enable easy configuration of application-specific scheduling policies. In addition, their sophisticated scheduling algorithms can be relatively expensive to execute, as we discussed in the previous subsection. Furthermore, requests for resources that cannot be satisfied in their entirety can wait for long periods of time in wait queues, in many cases longer than the execution time itself. Figure 2 shows the high variance of the queue wait times and the large percentage of time jobs spent in the waiting queue as opposed to executing at SDSC in 2004/2005 [37]. The results from Figure 2 are typical to what can be found in many busy Grid sites. Notice the average queue wait time of about 7.6 hours, while the average job run time was only 1.8 hours. Others have also found that the average wait time of grid jobs is higher in practice than the predictions from simulation-based research. [71] Finally, while batch schedulers may support different queues and policies, the policies implemented in a particular

instantiation may not be optimized for the particular workload producing the jobs. For example, a scheduler may allow only a modest number of concurrent submissions for a single user.

To address these problems, we propose dynamic resource provisioning, in which we seek general techniques to acquire and/or release resources as application demand varies that would be transparent to the domain-specific application logic. We also seek to reduce the average queue wait times by amortizing high overhead of resource allocation over the execution of many tasks. In our preliminary studies [6, 11], microbenchmarks show that through dynamic resource provisioning, we can allocate resources on the order of 10s of seconds across multiple Grid sites and can reduce average queue wait times by up to 95% (effectively yielding queue wait times within 3% of ideal); furthermore, applications (executed by the Swift parallel programming system) reduce end-to-end run time of up to 90% for large-scale astronomy and medical applications, relative to versions that execute tasks via separate scheduler submissions.



	Queue Wait Time (sec)	Job Run Time (sec)	Queue Wait Time %	Job Run Time %
Minimum	0	0	0%	0%
Average	32,020	6,456	59%	41%
Median	597	697	738%	27%
Maximum	2,456,666	141,276	100%	100%
Standard Deviation	81,291	13,630	38%	38%

Figure 2: Summary of queue wait times and job run times regarding 96,089 jobs submitted between March 2004 and March 2005 to the San Diego Supercomputer Center (SDSC) DataStar

1.1.3 Data Caching

The data in Figure 1 were simplified by only investigating the effects of dispatch and execution throughputs on efficiency, not taking into consideration any data I/O. Many applications are composed of pieces that compute on some data, and hence it is worthwhile to look at a real production Grid, namely the TeraGrid (TG) distributed infrastructure [17].

The TG is the world's largest, most comprehensive distributed cyber-infrastructure for open scientific research. Currently, TG resources include more than 250 teraflops of computing capability (4.8K nodes with over 14K processors) and more than 1 petabyte of spinning disk and 30 petabytes of archival data storage, with rapid access and retrieval over high-performance networks (10~30 Gb/s links). Of the 4.8K nodes in the TG, 4.2K nodes have local dedicated disks that are underutilized (both in terms of storage and I/O bandwidth). With an average modest disk size of 67GB per node, this totals to 283TB of disk in addition to the 1PB of shared storage resources. The locally installed disks on most TG nodes have an aggregate theoretical peak throughput on the order of 4000+ Gb/s, in contrast with the shared storage resources that have an aggregate peak throughput on the order of 100 Gb/s. Figure 3 shows actual measurements of a modest 64 node cluster, and the difference in performance between the GPFS shared file system and the theoretical local disk (3.4 Gb/s as opposed to 66 Gb/s read performance, and 1Gb/s as opposed to 25 Gb/s read/write performance).

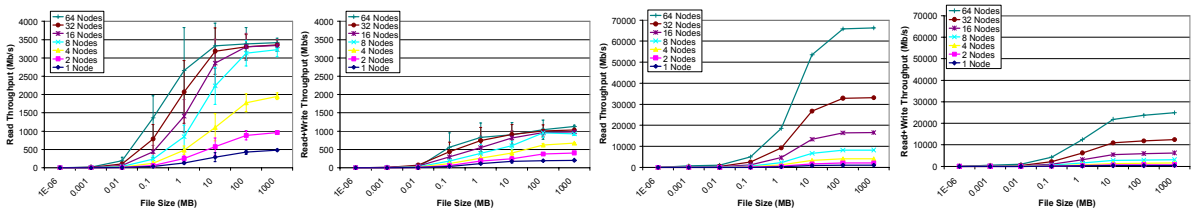


Figure 3: Read and read/write performance for GPFS (first two) and local disk (last two) expressed in Mb/s; only the x-axis is logarithmic; 1-64 nodes; 1B – 1GB files

We have experience with certain astronomy specific data access patterns on the TG that produce an order of magnitude difference in performance between processing data directly from local disk as opposed to accessing the data from shared storage resources (i.e. GPFS) [7, 8, 79, 94]. With more than an order of magnitude performance improvement when using local disks as well as better scalability with increasing number of compute resources, there is good motivation to have middleware capable of harnessing this relatively idle storage resource transparently for large scale scientific applications.

It has been argued that data intensive applications cannot be executed efficiently in grid environments because of the high costs of data movement. But if data analysis workloads have internal locality of reference [95], then it can be feasible to acquire and use even remote resources, as high initial data movement costs can be offset by many subsequent data analysis operations performed on that data. We can imagine data diffusing over an increasing number of CPUs as demand increases, and then contracting as load reduces.

We envision "data diffusion" as a process in which data is stochastically moving around in the system, and that different applications can reach a dynamic equilibrium this way. One can think of a thermodynamic analogy of an optimizing strategy, in terms of energy required to move data around ("potential wells") and a "temperature" representing random external perturbations ("job submissions") and system failures. We propose exactly such a stochastic optimizer.

1.2 Hypothesis

The analysis of large datasets typically follows a split/merge pattern, which includes an analysis query to be answered, which get split down into independent tasks to be computed, after which the results from all the tasks are merged back into a single aggregated result. The hypothesis is that significant performance improvements can be obtained in the analysis of large dataset by leveraging information about data analysis workloads rather than individual data analysis tasks. We define workloads to be a complex query that can be decomposed into simpler tasks, or a set of queries that together answer some broader analysis questions. As the size of scientific data sets and the resources required for analysis increase, data locality becomes crucial to the efficient use of large scale distributed systems for scientific and data-intensive applications [9]. We believe it is feasible to allocate large-scale computational resources and caching storage resources that are relatively remote from the original data location, co-scheduled together to optimize the performance of entire data analysis workloads.

1.3 Proposal

In order to support interactive analysis of large quantities of data, we propose a *data diffusion* approach that leverages Grid [22] infrastructures to acquire resources required for data analysis dynamically, in response to demand. As request rates increase, more resources are acquired, and data and applications diffuse from low-cost archival storage to newly acquired resources. Acquired resources can be "cached" for some time, thus allowing more rapid responses to subsequent requests. If demand drops, resources can be released. In principle, this approach can provide the benefits of dedicated hardware without the associated high costs—depending crucially on the nature of application workloads and the performance characteristics of the underlying infrastructure.

In order to explore the split/merge pattern introduced in the hypothesis combined with the proposed data diffusion, we define AMDASK, an Abstract Model for DAta-centric taSK farms. A data-centric task farm is a common parallel pattern that drives the independent computational tasks, taking into consideration the data locality in order to optimize the performance of the analysis of large datasets. This definition implies the integration of data semantics and application behavior to address critical challenges in the management of large scale datasets and the efficient execution of application workloads.

In order to gain experience with the practical realization of AMDASK, we are developing a Fast and Light-weight tasK executiON framework (Falkon) [6], which provides for dynamic acquisition and

release of resources ("executors") [11], data management capabilities [78, 79], and the dispatch of analysis tasks to those executors using various schedulers, including a data-aware scheduler. Furthermore, we intend to validate the AMDASK model through discrete-event simulations.

We have integrated Falkon into the Swift parallel programming system [13, 19, 88] in order to leverage the large number of Swift-based applications, which can be executed over Falkon transparently. We will investigate the effectiveness and the flexibility of the AMDASK model on applications from various domains (astronomy [7, 8, 12], astro-physics [40], medicine [32], chemistry, and economics), covering a variety of different datasets, workloads, and analysis codes. We believe AMDASK is an important model that generalizes to many domains and applications, and could offer AMDASK-based systems and applications end-to-end performance improvements and higher resource utilization and efficiency.

We intend to validate the AMDASK model more generally through simulations. We will implement the AMDASK model in a discrete event simulation that will allow us to investigate a wider parameter space than we could in a real world implementation and deployment. We expect the simulations to help us prove that the AMDASK model is both efficient and scalable given a wide range of simulation parameters (i.e. number of storage and computational resources, communication costs, management overhead, and workloads – including inter-arrival rates, query complexity, and data locality). The outputs from the simulations over the entire considered parameter space will form the datasets that will be used to statistically validate the model using R^2 *statistic* and g*raphical residual analysis.* [72]

1.4 Contributions

We see the dynamic analysis of large datasets to be important due to the ever growing datasets that need to be accessed by larger and larger communities. Attempting to address the storage and computational problems separately (essentially forcing much data movement between computational and storage resources) will not scale to tomorrow's peta-scale datasets and will likely yield significant underutilization of the raw computational resources. We defined the abstract model for data-centric task farms (AMDASK) in order to address the integration of the storage and computational issues found in a class of applications which can be decomposed down into many independent computational tasks that operate on large datasets. We prove the abstract model to be efficient and scalable through simulations. Finally, we provide a reference implementation of the abstract model which is used to show the flexibility and effectiveness of it on real world applications.

2 AMDASK: an Abstract Model for DAta-centric taSK farms

Many analyses of large datasets follow a split/merge methodology, which includes an analysis query to be answered, the splitting down into independent tasks to be computed, and the merging of independent results into a single aggregated result. Based on this split/merge pattern, we propose AMDASK, an Abstract Model for DAta-centric taSK farms. The literature defines task farms as a common parallel pattern which drives the computation of independent tasks, where a task is a self contained computation. The data-centric component of the abstract model emphasizes the central role data plays in the task farm model we are proposing, and the fact that the task farm is optimized to take advantage of data cache storage and data locality found in many large datasets and typical application workloads. We define a data-centric task farm as a parallel pattern that drives the independent computational tasks taking into consideration data locality to optimize the performance of the analysis of large datasets. This definition implies the integration of data semantics and application behavior in order to address critical challenges in the management of large scale datasets and the efficient execution of application workloads.

The rest of this section covers the formal definition of the proposed abstract mode, the execution model, and the performance and efficiency of the defined model.

2.1 Base Definitions and Notations

A data-centric task farm has various components (i.e. computational resource where the tasks are to execute, storage resources where the data needed by the tasks is stored, etc) that we will formally define in this sub-section. Since our model is a data-centric one, we assume that the computational tasks will need to work on some data.

Persistent data stores: To simplify the model, we assume that this data resides on a *set of persistent data stores*, Π, where $|\Pi| \geq 1$. This is a realistic assumption as most large datasets have a long lifespan due to various factors; these factors include 1) the cost of acquiring them, 2) the time needed to analyze the entire dataset by entire communities, and 3) the need to keep these datasets online for legacy applications and the repeatability of results after they have been completed.

Transient data stores: We differentiate between the set of persistent data stores Π and the *set of transient data stores*, T, as they have different characteristics (performance, availability, scalability, capacity, etc.). For example, persistent data stores could be shared among many users and applications, and hence should offer improved scalability and performance; however, persistent data stores do not have computational resources available tightly coupled with the storage resource, and hence the latency incurred between the computational resource and the persistent storage resource is higher. Furthermore, persistent data stores are expected to be always available, which is achieved by data store backups that could be brought online if the primary persistent data store becomes unavailable. The set of transient data stores T, where $|T| \geq 0$, are smaller than the persistent data stores and are only capable of storing a fraction of the persistent data stores' data objects.

Transient resources: We assume that the transient data stores T are co-located with compute resources, hence yielding a lower latency data path than the persistent data stores. Since storage and compute resources are co-located, we refine the definition of T to include computational resources as well. This definition of a transient data store and computational resource is consistent with real world distributed systems that have a pool of compute resources that are allocated and de-allocated by a resource manager based on some policies and requirements, hence the transient nature of the resources. The co-location of the storage resource and the computational resource is also consistent with real world system as most computational resources have attached storage resources as well. The relationship between the *transient resources* and the persistent data stores can be summarized by $T \subseteq \Pi$; note that the reverse is not always true, as storage resources do not always have computational resources, which is why the persistent data store does not have a persistent computational resource co-located with it.

Data Objects: $\phi(\pi)$ represents the *data objects found in the persistent data store* π, where $\pi \in \Pi$. Similarly, $\phi(\tau)$ represents a *transient data store's locally cached data objects*. The set of persistent data stores Π consists of a *set of all data objects*, Δ. For each *data object* $\delta \in \Delta$, $\beta(\delta)$ denotes the *data object's size* and $\lambda(\delta)$ denotes the *data object's storage location(s)*.

Store Capacity: For each persistent data store, $\pi \in \Pi$, and transient data store $\tau \in T$, $\sigma(\pi)$ and $\sigma(\tau)$ denote the persistent and transient *data store's capacity*, respectively.

Compute Speed: For each transient resource, $\tau \in T$, $\chi(\tau)$ denotes the *compute speed*.

Load: For any data store, we define *load* as being the number of concurrent read/write requests; $\omega(\tau)$ and $\omega(\pi)$ denote the load on data stores $\tau \in T$ and $\pi \in \Pi$, respectively.

Ideal Bandwidth: For any persistent data store $\pi \in \Pi$, and transient data store $\tau \in T$, $\nu(\pi)$ and $\nu(\tau)$ denote the *ideal bandwidth for the persistent and transient data store*, respectively. These transient data stores will have limited availability, and the bandwidth will typically be lesser when compared to the persistent data stores, $\nu(\tau) < \nu(\pi)$. The organization in real world deployments of persistent and transient

data stores is that there are few high capacity persistent data stores and many low capacity transient data stores. Thus, we assume $\sum_{\tau \in T} \nu(\tau) \geq \sum_{\pi \in \Pi} \nu(\pi)$, given that $|T| >> |\Pi|$.

Available Bandwidth: For any persistent data store $\pi \in \Pi$, and transient data store $\tau \in T$, we define *available bandwidth* as a function of ideal bandwidth and load; more formally, $\eta(\nu(\pi), \omega(\pi))$ and $\eta(\nu(\tau), \omega(\tau))$ will denote the available bandwidth for the persistent and transient data store, respectively. The relationship between the ideal and available bandwidth is given by the following formula: $\eta(\nu(\pi), \omega(\pi)) < \nu(\pi)$, for $\omega(\pi) \geq 1$ and $\eta(\nu(\pi), \omega(\pi)) = \nu(\pi)$, for $\omega(\pi) = 0$.

Copy Time: For any data object $\delta \in \Delta$ and transient data store $\tau \in T$, we define the *time to copy a data object* between the object δ to τ by the function $\zeta(\delta, \tau) = \begin{cases} \tau_1 \to \tau, \forall \delta \in \phi(\tau_1), \tau_1 \in T \\ \pi_1 \to \tau, \forall \delta \in \phi(\pi_1), \pi_1 \in \Pi \setminus T \end{cases}$, where $\tau_1, \pi_1 \to \tau$ denotes the source and destination data stores for the copy operation. In an ideal case, $\tau_1 \to \tau$ can be computed by $\frac{\min[\nu(\tau_1), \nu(\tau)]}{\beta(\delta)}$, where $\nu(\tau_1)$ and $\nu(\tau)$ represent the source and destination *ideal bandwidth*, respectively, and $\beta(\delta)$ represents the data object's size; the same definition applies to copy a data object from $\pi_1 \to \tau$. In reality, this is an oversimplification since copy time $\zeta(\delta, \tau)$ is dependent on other factors such as the load $\omega(\tau)$ on some storage resource, the latency between the source and destination, and the error rates encountered during the transmission. Assuming low error rates and low latency (typical of a local area network environment), the copy time is then affected only by the data object's size and the available bandwidth $\eta(\nu(\tau), \omega(\tau))$ as defined above. More formally, in the realistic approach, we define $\tau_1 \to \tau$ as being $\frac{\min[\eta(\nu(\tau_1), \omega(\tau_1)), \eta(\nu(\tau), \omega(\tau))]}{\beta(\delta)}$.

Tasks: Let K denote the *incoming stream of tasks*. For each *task* $\kappa \in K$, let $\mu(\kappa)$ denote the *time needed to execute the task* κ on the computational resource $\tau \in T$; let $\theta(\kappa)$ denote the *set of data objects that the task* κ *requires*, $\theta(\kappa) \subseteq \Delta$; let $o(\kappa)$ denote the *time to dispatch the task* κ *and return a result*. In reality, there would also be some computational time needed to aggregate the results from various tasks, but for simplicity and to make the tasks truly independent as the definition of task farms dictates, we assume the time to aggregate the results is 0.

Computational Resource State: If a compute resource $\tau \in T$ is computing a task, then it is in the *busy* state, denoted by τ_b; otherwise, it is in the free state, τ_f. Let T_b denote the set of all compute resources in the busy state, and T_f the set of all compute resources in the free state; these two sets have the following property: $T_b \cup T_f = T$.

2.2 The Execution Model

The execution model outlines the respective policies that control various parts of the execution model and how they relate to the definitions in the previous section. Each incoming task $\kappa \in K$ is dispatched to a transient resource $\tau \in T$, selected according to the *dispatch policy* discussed in the following sections. If a response is not received after a time determined by the *replay policy*, or a failed response is received, the task is re-dispatched according to the *dispatch policy*. A missing data object, $\delta \in \Delta$, that is required by task κ, $\delta \in \theta(\kappa)$, and does not exist on the transient data store $\tau \in T$, $\delta \notin \phi(\tau)$, is copied from transient or persistent data stores selected according to the *data fetch policy*. If necessary, existing data at a transient data store τ are discarded to make room for the new data, according to the *cache eviction policy*. Each computation κ is performed on the data objects $\phi(\tau)$ found in a transient data store. When all

computations are complete, the result is aggregated and returned; this aggregation of the results is assumed to be free to simplify the abstract model.

We have repeatedly made the distinction between persistent data stores and transient storage/computational resources. The transient resources are an artifact of real world systems that are organized in such a fashion that resources are allocated and de-allocated from a pool of resources by some manager. This manager is basically coordinates the sharing of these resources among many consumers. We define a *resource acquisition policy* that decides when, how many, and for how long to acquire new transient computational and storage resources for. Similarly, we also define a *resource release policy* that decides when to release some acquired resources.

2.2.1 Dispatch Policy

Each incoming task $\kappa \in \mathrm{K}$ is dispatched to a transient resource $\tau \in \mathrm{T}$, selected according to the *dispatch policy*. We define four dispatch policies: 1) *next-available,* 2) *first-available*, 3) *max-cache-hit*, and 4) *max-compute-util*.

The **next-available** policy ignores any data location information when selecting an executor for a task; it simply chooses the next available executor. Thus, all data needed by a task must be transferred from the globally accessible storage resource.

The **first-available** policy ignores data location information when selecting an executor for a task, but it does include data location information along with each task. Thus, all data needed by a task must, with high probability, be transferred from the globally accessible storage resource, or another storage resource that has the data cached.

The **max-cache-hit** policy uses information about data location to dispatch each task to the executor with the largest number of data needed by that task. If that executor is busy, task dispatch is delayed until the executor becomes available. This strategy can be expected to reduce data movement operations relative to first-available, but may lead to load imbalances, especially if data popularity is not uniform. This policy aims at maximizing the cache hit/miss ratio, where a cache is the same as the transient data store; a cache hit occurs when a transient compute resource has the needed data on the same transient data store, and a cache miss occurs when the needed data is not the same computational resource's data store.

Formally, we define a *cache hit* as follows:

$\forall \delta \in \theta(\kappa), \exists \tau \in \mathrm{T}$, such that $\delta \in \phi(\tau)$.

Similarly, we define a *cache miss* as follows:

$\exists \delta \in \theta(\kappa)$, such that $\forall \tau \in \mathrm{T}, \delta \notin \phi(\tau)$.

Let $C_h(\kappa)$ denote the set of all cache hits, and $C_m(\kappa)$ denote the set of all cache misses for task $\kappa \in \mathrm{K}$, such that $C_h(\kappa) \bigcup C_m(\kappa) = \theta(\kappa)$.

We define the *max-cache-hit* dispatch policy as follows:

$$\max_{\forall \kappa \in \mathrm{K}} \left(\frac{C_h(\kappa)}{C_m(\kappa)} \right).$$

Note that this policy will cause the dispatcher to wait for compute resources to become free if all the compute resources that have the needed cached data are busy.

The **max-compute-util** policy also leverages data location information, but in a different way. It always sends a task to an available executor, but if several workers are available, it selects that one that has the most data needed by the task. This policy aims to maximize computational resource utilization.

We define a *free cache hit* as follows:

$\forall \delta \in \theta(\kappa), \exists \tau \in \mathrm{T}_f$, such that $\delta \in \phi(\tau)$.

Similarly, we define a *free cache miss* as follows:

$\forall \tau \in \mathrm{T}, \delta \notin \phi(\tau)$ or $\exists \tau \in \mathrm{T}_b$, such that $\delta \in \phi(\tau)$.

Let $C_{f,h}(\kappa)$ denote the set of all free cache hits, and $C_{f,m}(\kappa)$ denote the set of all free cache misses for task $\kappa \in \mathrm{K}$, such that $C_{f,h}(\kappa) \subseteq C_h(\kappa)$ and $C_m(\kappa) \subseteq C_{f,m}(\kappa)$ and $C_{f,h}(\kappa) \bigcup C_{f,m}(\kappa) = \theta(\kappa)$.

We define the *max-compute-util* dispatch policy as follows:

$$\max_{\forall \kappa \in \mathrm{K}} \left(\frac{C_{f,h}(\kappa)}{C_{f,m}(\kappa)} \right).$$

This policy would always send a task to any free computational resource, giving preferences to the computational resources that have the needed data objects in their data stores.

2.2.2 Replay Policy

The *replay policy* is fairly straight forward, but it is needed to ensure the correctness of the abstract model in realistic settings. It would determine the time to wait after the dispatch policy had been invoked and before determining to invoke the replay policy. It would also dispatch a failed task according to the dispatch policy.

2.2.3 Data Fetch Policy

There are *two data fetch policies* we define: 1) Just-in-Time Pre-Fetching and 2) Spatial Locality Pre-Fetching.

Just-in-Time Pre-Fetching: This is the simplest data fetch policy invoked by the transient data store τ to copy the needed data objects δ to the local data store. Formally, if $\delta \notin \phi(\tau)$ then $\zeta(\delta, \tau)$.

Spatial Locality Pre-Fetching: This data fetch policy builds on top of the copy the Just-in-Time Pre-Fetching in the sense that it attempts to populate transient data stores with data objects that are likely to be needed in the future. *Spatial locality pre-fetching* targets those applications access patterns that request a contiguous set of data in a spatial coordinate system. Assuming that the incoming tasks are processing data following this pattern, a needed data object δ can act as a good indicator to a future needed data object δ_1, which could be brought into transient data stores before it is actually needed with the same policy as the Just-in-Time Pre-Fetching.

Formally, if $\delta \notin \phi(\tau)$ then [$\zeta(\delta, \tau)$ and if $\delta_1 \notin \phi(\tau)$ then $\zeta(\delta_1, \tau)$, $\forall \delta_1 \in \mathrm{N}(\delta)$], where $\mathrm{N}(\delta)$ is the neighborhood of δ in the spatial domain.

2.2.4 Cache Eviction Policy

The ***cache eviction policies*** outlined here are complete set of supported policies, however it is unclear which policies will be better suited for certain workloads and applications. We define a cache eviction policy to be the selection of an existing data object in a transient data store to be removed in order to make room for a new data object to be inserted in the same data store. We explicitly define five classes of cache eviction policies: 1) Random, 2) First-In-First-Out, 3) Least Recently Used (LRU, LRU2, and Two Queues), 4) Least Frequently Used (Perfect, In Cache, and Hierarchical), and 5) Time-based Expiration (Simple, Extended, and Sliding). Since these caching policies are typical caching schemes found in the literature [15], we will not go into the details of defining each one. Note that items and data objects, as well as transient data stores and caches, are both used interchangeably in this section.

2.2.5 Resource Acquisition Policy

We define a *resource acquisition policy* that decides when and how to acquire new transient computational and storage resources. The resource acquisition policy should determine the state information that will be used to trigger new computational resource acquisitions to be added to T (i.e. if the task queue length increases, acquire more resources). It should also determine the number of nodes to acquire based on the appropriate state information. The policy would also need to determine the length of time to acquire the resources for based on the state information.

This policy determines the number of resources, *n*, to acquire; the length of time for which resources should be requested; and the request(s) to generate to LRM(s) to acquire those resources. We have implemented five strategies that variously generate a single request for *n* resources, *n* requests for a single resource, or a series of arithmetically or exponentially larger requests, or that use system functions to determine available resources.

2.2.6 Resource Release Policy

Just as we defined a resource acquisition policy, we define a *resource release policy* that decides when to release some already acquired resources. We define two such policies: 1) Centralized, and 2) Distributed. In a *centralized* policy, decisions are made based on state information available at a central location. For example: "if there are no queued tasks, release all resources," and "if the number of queued tasks is less than *q*, release a resource." In a *distributed* policy, decisions are made at individual resources based on state information available at the resource. For example: "if the resource has been idle for time *t*, the resource should release itself." Note that resource acquisition and release policies are not independent: in most batch schedulers, a set of resources allocated in a single request must all be de-allocated before the requested resources become free and ready to be used by the next allocation. Ideally, one must release all resources obtained in a single request at once, which requires a certain level of synchronization among the resources allocated within a single allocation.

2.3 The Performance and Efficiency of the Abstract Model

In this section, we investigate when we can achieve good performance with this abstract model for data-centric task farms and under what assumptions. We define various costs (costs per task and average task execution time) and efficiency related metrics (efficiency, computational intensity, efficiency overheads). Furthermore, we explore the relationships between the different parameters in order to optimize efficiency.

Cost per task: For simplicity, let us assume initially that each task requires a single data object, $\delta \in \theta(\kappa)$, and that all the data objects Δ on persistent storage Π and transient storage T are fixed; assume that we use the max-resource-util dispatch policy, in which we send a task to a compute resource that contains the required data object, or to any other free resource otherwise. Then the *cost of the execution of each task* $\kappa \in \mathrm{K}$ dispatched to a transient compute resource $\tau \in \mathrm{T}$ can be characterized as one of the following two costs: 1) *cost if the required data objects are cached* at the corresponding transient storage resource, and 2) *cost if the required data objects are not cached* and must be retrieved from another transient or persistent data store. In the first case, we define the cost of the execution of a task to be the time to dispatch the task plus the time to execute the task plus the time to return the result. For the second cost function in which the data objects do not exist in the transient data store, we also incur an additional cost to copy the needed data object from either a persistent or a transient data store. More formally, we define the *cost per task* $\chi(\kappa)$ as follows:

$$\chi(\kappa) = \begin{cases} o(\kappa) + \mu(\kappa), & \delta \in \phi(\tau) \\ o(\kappa) + \mu(\kappa) + \zeta(\delta, \tau), & \delta \notin \phi(\tau) \end{cases}$$

Average Task Execution Time: We define the *average task execution time,* B, as the summation of all the task execution times divided by the number of tasks; more formally, we have $\mathrm{B} = \frac{1}{|\mathrm{K}|}\sum_{k\in\mathrm{K}}\mu(\kappa)$.

Computational Intensity: Let A denote the *arrival rate of tasks*; we define the *computational intensity,* I, as follows: $\mathrm{I} = \mathrm{B}*\mathrm{A}$. If $\mathrm{I}=1$, then all nodes are fully utilized; if $\mathrm{I}>1$, tasks are arriving faster than they can be executed; finally, if $\mathrm{I}<1$, then there are nodes that might be idle.

Workload Execution Time: We define the *workload execution time,* V, of our system as $V = \max\left(\frac{B}{|\mathrm{T}|}, \frac{1}{\mathrm{A}}\right)*|\mathrm{K}|$.

Workload Execution Time with Overhead: In general, the total execution time for a task $\kappa\in\mathrm{K}$ includes overheads, which reduced efficiency by a factor of $\frac{\mu(\kappa)}{\chi(\kappa)}$. We define the *workload execution time with overhead,* W, of our system as $W = \max\left(\frac{\mathrm{Y}}{|\mathrm{T}|}, \frac{1}{\mathrm{A}}\right)*|\mathrm{K}|$, where Y is the *average task execution time including overheads* defined as $Y = \begin{cases} \frac{1}{|\mathrm{K}|}\sum_{\kappa\in\mathrm{K}}[\mu(\kappa)+o(\kappa)], & \delta\in\phi(\tau),\delta\in\Omega \\ \frac{1}{|\mathrm{K}|}\sum_{\kappa\in\mathrm{K}}[\mu(\kappa)+o(\kappa)+\zeta(\delta,\tau)], & \delta\notin\phi(\tau),\delta\in\Omega \end{cases}$.

Efficiency: We define the *efficiency,* E, of a particular workload as $\mathrm{E} = \frac{V}{W}$. The expanded version of efficiency is $\mathrm{E} = \frac{\max\left(\frac{B}{|\mathrm{T}|}, \frac{1}{\mathrm{A}}\right)*|\mathrm{K}|}{\max\left(\frac{\mathrm{Y}}{|\mathrm{T}|}, \frac{1}{\mathrm{A}}\right)*|\mathrm{K}|}$, which can be reduced to $E = \begin{cases} 1, & \frac{Y}{|T|}\le\frac{1}{A} \\ \max\left(\frac{B}{Y}, \frac{|\mathrm{T}|}{\mathrm{A}*Y}\right), & \frac{Y}{|T|}>\frac{1}{A} \end{cases}$.

We claim that for the caching mechanisms to be effective in this model (i.e. the needed data objects to be found in transient data stores), the *aggregate capacity of our transient storage resources* T *is greater than our workload's working set,* Ω, (all data objects required by a sequence of tasks) *size*; formally, we can say $\sum_{\tau\in\mathrm{T}}\sigma(\tau)\ge|\Omega|$.

We also claim that we can obtain $\mathrm{E}>0.5$ if $\mu(\kappa)>o(\kappa)+\zeta(\delta,\tau)$, where $\mu(\kappa)$, $o(\kappa)$, $\zeta(\delta,\tau)$ are the time to execute and dispatch the task $\kappa\in\mathrm{K}$, and copy the object δ to $\tau\in\mathrm{T}$, respectively.

Speedup: We define the *speedup,* S, of a particular workload as $S = E*|T|$.

Optimizing Efficiency: Having defined both efficiency and speedup, it is possible to maximize for either one, as efficiency normally monotonically decreases and speedup increases with more resources used. We can *optimize efficiency* by finding the smallest number of *transient compute/storage resources* $|\mathrm{T}|$ while we maximize speedup times efficiency.

3 Simulations

We will validate the AMDASK model through simulations, to verify the both the efficient and scalability given a wide range of simulation parameters (i.e. number of storage and computational resources, communication costs, management overhead, and workloads – including inter-arrival rates, query complexity, and data locality). We will implement the model in a discrete event simulation that will allow us to investigate a wider parameter space than we could in a real world implementation.

The simulations will specifically attempt to model a grid computing environment comprising of computational resources, storage resources, batch schedulers, various communication technologies, various types of applications, and workload models. We will perform careful and extensive empirical performance evaluations in order to create correct and accurate input models to the simulator; the input models include 1) Communication costs, 2) Data management costs, 3) Task scheduling costs, 4) Storage access costs, and 5) Workload models.

We expect to be able to scale simulations to more computational and storage resources than we could achieve in a real deployed system due to the availability of resources. Furthermore, assuming the input models to be correct, we should be able to accurately measure the end-to-end performance of various applications using a wide range of strategies for the various resource management components.

Model Validation: The outputs from the simulations over the entire considered parameter space will form the datasets that will be used to statistically validate the model using R^2 *statistic* and g*raphical residual analysis*.

R^2 **Statistic:** It measures the fraction of the total variability between the model's estimated values and the simulated derived values. The R^2 **Statistic** is the square of the correlation coefficient which is calculated as follows: $corr(e_i, \mathrm{E}_i) = \frac{1}{n-1}\sum_{i=1}^{n}\left[\frac{e_i - \bar{e}_i}{std(e_i)} * \frac{E_i - \bar{E}_i}{std(E_i)}\right]$. A value close to one denotes that most of the variability in the data is captured by the proposed model.

Residual Analysis: Often, a high R^2 value does not guarantee that the model fits the data well. Use of a model that does not fit the data well cannot provide good answers to the underlying abstract model and will likely misrepresent the behavior of the model in practice. There are many statistical tools for model validation, but the primary tool for most process modeling applications is graphical residual analysis. Different types of plots of the residuals from a fitted model provide information on the adequacy of different aspects of the model. Graphical methods have an advantage over numerical methods (i.e. R^2 statistic) for model validation because they readily illustrate a broad range of complex aspects of the relationship between the model and the data. Numerical methods for model validation tend to be narrowly focused on a particular aspect of the relationship between the model and the data and often try to compress that information into a single descriptive number or test result.

The residuals from a fitted model are the differences between the responses observed at each combination values of the explanatory variables and the corresponding prediction of the response computed using the regression function. In the context of our proposed model, the definition of the residual for the i^{th} observation in the data set is $r_i = e_i - E_i$ with e_i denoting the i^{th} efficiency response in the simulated data set and E_i is the efficiency as calculated by the abstract model.

If the model fit to the data were correct, the residuals would approximate the random errors that make the relationship between the explanatory variables and the response variable a statistical relationship. Therefore, if the residuals appear to behave randomly, it suggests that the model fits the data well. On the other hand, if non-random structure is evident in the residuals, it is a clear sign that the model fits the data poorly. [72]

4 Related Work

Our proposed work covers a wide range of research topics and areas. We believe it is important to discuss other work in the literature that are related to our proposal. For a more coherent organizational structure, the related work is partitioned into several categories: 1) data-centric task farms, 2) grid simulations, 3) resource management, and 4) applications.

4.1 Task Farms

Task farming is a general concept that has been applied on a wide range of systems. The Blue Gene supercomputer is one example which has defined and implemented task farms in order to implement parallelism in some applications. [49] Casanova et al. addresses basic scheduling strategies for task farming in Grids [47]; they acknowledge the difficulties that arise in scheduling task farms in dynamic and heterogeneous systems, but do little to address these problems. M. Danelutto argues the inefficiencies of task farms in heterogeneous and unreliable environments; he proposes various adaptive task farm implementation strategies [44] to address these classical inefficiencies found in task farms. H. Gonzalez-Velez argues for similar inefficiencies for task farms due to heterogeneity typically found in Grids. He claims that the dynamicity of Grids also leads to sub-optimal task farm solutions. He proposes an adaptive skeletal task farm for Grids [46] which take into account predictions of network bandwidth, network latency, and processor availability. Heymann et al. investigates scheduling strategies that dynamically measures the execution times of tasks and uses this information to dynamically adjust the number of workers to achieve a desirable efficiency, minimizing the impact in loss of speedup. [45] Petrou et al. show how scheduling speculative tasks in a compute farm [48] can significantly reduce the visible response time. The basic idea is that a typical end use would submit more work than he really needed in the hopes of allowing the scheduler ample opportunities to schedule work before the end user needed to retrieve the results. We believe that this model of scheduling would work only in a lightly loaded compute farm, which is not the norm in today's deployed Grids. In summary, all the related work we found that targeted task farms and scheduling did not address the "data-centric" part of our task farm model. Heterogeneity and dynamicity of Grids is indeed a problem for task farms, however unless careful consideration is given to the data that the task farm must operate on, it is unlikely that the raw compute and storage resources are utilized as efficiently as they could be.

4.2 Grid Simulations

There has been much work in Grid simulations, mostly due to the advantages they offer in easily testing different scheduling algorithms, data management strategies, replication strategies, etc on large scale Grids without the need to actually have access to large amounts of resources. The simulations that are most closely related to the work we intend to pursue in order to validate the AMDASK model revolve around scheduling of jobs or tasks in relation to the data management strategies. We intend to use the GridSim simulator [41, 42, 43] implemented as part of the GridBus project. The same group also made extensions to GridSim that allows the modeling and simulation of data grids with integration of data storage, replication and analysis [56]. These extensions could prove to be invaluable to our own simulation work in validating the AMDASK model. The same group also has done work in the dynamic job grouping-based scheduling for deploying applications with fine-grained tasks on global Grids [57]; the scheduling is done based on the task granularity, a concept that is important in the AMDASK model, and that we plan to explore through both simulations and in the real implemented systems. Another simulator which addresses both data management and scheduling is OptorSim [53]. It was designed to investigate Grid environments for studying dynamic data replication strategies. The authors show the promising advantages of data replication based on usage patterns for the Large Hadron Collider experiments at CERN. [54, 55] A similar simulation study, ChicSim, shows the effects of data replication on the scheduling strategies. [58] The experiments differ from our proposed work in that the replication strategies assume the replication is taking place between the well connected storage resources at different sites, and do not take into consideration the fact that computational nodes also have attached storage as

we do. Finally, Singh et al. investigates the performance impact of resource provisioning on workflows [25], and shows through simulations that considerable performance improvements can be obtained from resource provisioning. We plan to pursue this study in both simulations and in practice.

4.3 Resource Management

Task Dispatch: Full-featured local resource managers (LRMs) such as Condor [18], Condor-J2 [27], PBS [28], LSF [29] support client specification of resource requirements, data staging, process migration, check-pointing, accounting, and daemon fault recovery. In contrast, our own work Falkon is not a full-featured LRM: it focuses solely on efficient task dispatch and thus can omit some of these features in order to streamline task submission. This narrow focus is possible because Falkon can rely on LRMs for certain functions (e.g., accounting) and clients for others (e.g., recovery, data staging). The BOINC "volunteer computing" system [30, 31] has a similar architecture to that of Falkon. However, it is not clear how much BOINC would need to adapt in order to be used in production Grids, the main focus and aim of our work. In summary, Falkon's innovation is its combination of a fast lightweight scheduling overlay on top of virtual clusters with the use of standard grid protocols for adaptive resource allocation. This combination of techniques allows us to achieve higher task throughput than previous systems, while also offering applications the ability to trade-off system responsiveness, resource utilization, and execution efficiency.

Resource Provisioning: Multi-level scheduling has been applied at the OS level [27, 30] to provide faster scheduling for groups of tasks for a specific user or purpose by employing an overlay that does lightweight scheduling within a heavier-weight container of resources: e.g., threads within a process or pre-allocated thread group. Frey and his colleagues pioneered the application of this principle to clusters via their work on Condor "glide-ins" [23]. Requests to a batch scheduler (submitted, for example, via Globus GRAM4 [36]) create Condor "startd" processes, which then register with a Condor resource manager that runs independently of the batch scheduler. Others have also used this technique. For example, Mehta et al. [26] embed a Condor pool in a batch-scheduled cluster, while MyCluster [24] creates "personal clusters" running Condor or SGE. Such "virtual clusters" can be dedicated to a single workload; thus, Singh et al. find, in a simulation study [25], a reduction of about 50% in completion time, due to reduction in queue wait time. However, because they rely on heavyweight schedulers to dispatch work to the virtual cluster, the per-task dispatch time remains high. Appleby et al. [33] were one of several groups to explore *dynamic* resource provisioning within a data center. Ramakrishnan et al. [34] also address adaptive resource provisioning with a focus primarily on resource sharing and container level resource management. Our work differs in its focus on performing resource provisioning on non-dedicated resources that are managed by LRMs. J. Bresnahan addresses resource provisioning through an architecture for dynamic allocation of compute cluster bandwidth [35], in which he modified the Globus GridFTP server to support the dynamic allocation of additional GridFTP servers under load. Of all the resource provisioning work, only the dynamic GridFTP server work does a dynamic resizing of the resource pool based on some metrics, such as the load on the system.

Data Management: The Globus Toolkit includes two components (Replica Location Service [59] and Data Replication Service [60]) that can be used to build data management services for Grid environments. Several large projects (Mobius [63] and ATLAS [61]) implemented their own data management systems to aid in the respective application's implementations. Google also has their own data management system called BigTable, a distributed storage system for structured data [68]. Finally, GFarm is a Grid file system that supports high-performance distributed and parallel data computing [50]. Yamamoto also explores the use of GFarm for petascale data intensive computing [51].

Data management on its own is useful, but not as useful as it could be if it were to be coupled with compute resource management as well. Ranganathan et al. performed simulation studies of computation and data Scheduling algorithms for data Grids [58]. The GFarm team implemented a data aware scheduler in GFarm using LSF scheduler plug-in mechanism [2]. Finally, from Google, the combination of

BigTable, Google File System (GFS) [5], and MapReduce [70] yields a system that could potentially have all the advantages of our proposed data-centric task farms, in which computations and data are co-located on the same resources. Although both Google and GFarm address the coupling of data and computational resources, they both assume a relatively static set of resources, one that is not commonly found in today's Grids. In our work, we further extend this fusion of data and compute resource management by also enabling dynamic resource provisioning, which essentially allows our solution to operate over today's batch-based Grids. It is worth noting that both Google's approach and GFarm require dedicated resources, which makes the problem more tractable when compared to our solution which assumes that the resources are shared among many users through batch-scheduled systems, which leases some set of resources for finite periods of time.

4.4 Analysis of Large Datasets

All our work is motivated by the potential to improve application performance and even enable the ease of implementation of certain applications that would otherwise be difficult to implement with adequate performance for the applications to be useful. This sub-section covers an overview of a broad range of systems used to perform analysis on large datasets. The DIAL project that is part of the PPDG/ATLAS project focuses on the distributed interactive analysis of large datasets [66]. Google has combined several of their projects such as BigTable [68], Sawzall [69], MapReduce [70], and GFS [5] in order to address some of Google biggest data analysis challenges in indexing the world wide web. Chervenak et al. developed the Earth System Grid-I prototype to analyze climate simulation data using data Grid technologies [64]. As we previously discussed, the Mobius project developed a sub-project DataCutter for distributed processing of large datasets [65]. A database oriented view for data analysis is taken in the design of GridDB, a data-centric overlay for the scientific Grid [67]. Finally, Olson et al. discusses Grid service requirements for interactive analysis of large datasets [62]. All in all, what all these projects lack is either co-location of storage and computations close to each other (i.e. on the same physical resource), or they lack the assumption that Grid systems are managed by batch schedulers which complicates the deployment of permanent data management infrastructure such as Google's GFS or the GFarm file system.

5 Completed Milestones and Future Work

In order to put in context the proposed work, it is worthwhile to discuss the current state of the work, and what has been accomplished to date. This section will discuss the completed milestones followed by future work.

5.1 Completed Milestones

There are various fundamental research questions we hope to address through our work presented in this proposal. They center on two main areas, data and compute resource management, and how they relate to particular workloads of data analysis on large datasets. We have completed several key milestones which have set the groundwork for the near-term future work. These milestones are in four main areas: 1) defining an abstract task farm model, 2) resource management, 3) data diffusion, and 4) applications.

Abstract task farm model: In order to explore the split/merge methodology found in many applications, combined with the proposed data diffusion, we have formally defined AMDASK, an Abstract Model for DAta-centric taSK farms. A data-centric task farm is defined as a common parallel pattern that drives the independent computational tasks, taking into consideration the data locality in order to optimize the performance of the analysis of large datasets.

Resource Management: To enable the rapid execution of many tasks on compute clusters, we have developed Falkon, a Fast and Light-weight tasK executiON framework. [6] In essence, Falkon is the practical realization of the abstract model AMDASK. Falkon integrates (1) multi-level scheduling to separate resource acquisition (via, e.g., requests to batch schedulers) from task dispatch, and (2) a

streamlined dispatcher. Falkon's integration of multi-level scheduling and streamlined dispatchers delivers performance not provided by any other system. Microbenchmarks show that Falkon throughput (between 200 to 5000 tasks/sec depending on configuration and optimizations used) and scalability (to 54,000 executors and 1,000,000 tasks processed in just 400 seconds) are several orders of magnitude better than other systems used in production Grids. [6] Large-scale astronomy and medical applications executed under Falkon by the Swift parallel programming system [13, 19] achieve up to 90% reduction in end-to-end run time, relative to versions that execute tasks via separate scheduler submissions. [6, 13] Furthermore, the dynamic resource provisioning proposed in Falkon can allocate resources on the order of 10s of seconds across multiple Grid sites and can reduce average queue wait times by up to 95% (effectively yielding queue wait times within 3% of ideal). [11, 73] Falkon's latest stable release v0.9 has been tested in various environments, including the TeraGrid [17], TeraPort [74], and Tier3 [75]. We have also submitted a proposal to make Falkon a Globus incubator project and are waiting on feedback [93]. Finally, we have two separate initiatives that are still work in progress, but have already made some progress. The first is extending provisioning from batch-scheduled systems to economic ones, such as the Amazon Elastic Computing Cloud (EC2) [96]. The second is to move from a 2-Tier architecture in Falkon to a 3-Tier one, and to explore alternative technologies in order to be able to run Falkon on the IBM BlueGene Supercomputer [90].

Data Diffusion: We first identified that data locality is important in large scale scientific exploration [9]. We have developed a reference implementation [76, 77] of the data diffusion as part of the AstroPortal [76, 77], where the AstroPortal [7, 8] is an astronomy application that is tightly coupled to Falkon; we discuss this further in the following applications sub-section. The reference implementation performed all the basic requirements of the proposed data diffusion (caching of image data on local disk, data–aware scheduler) with some specific assumptions made due to the particular astronomy application and dataset we were using. We investigated the performance of the data-aware scheduler and found it do be sufficiently fast for medium size Grids, but did not scale well to large grids. [77] With the goal to generalize the data management implementation, we ported the reference implementation directly into Falkon for any other application that ran over Falkon to use transparently [78]. We have presented some preliminary results on the data diffusion component of Falkon at the 2007 Microsoft eScience Workshop at RENCI [79, 78]. We plan to update the Falkon provider code in Swift to enable the data diffusion to be used by any Swift application; this work is expected to be completed by 2/1/08.

Applications: We have integrated Falkon into the Karajan [19, 13] workflow engine, which in term is used by the Swift parallel programming system [13]. Thus, Karajan and Swift applications can use Falkon without modification, and hence we are able to leverage a wide range of scientific applications that are already implemented and run over Swift. We have tested several Swift-based applications from various domains including astronomy [6, 13, 88], medicine [6, 13, 88], chemistry [38], and economics [39] with varying datasets, workloads, and analysis codes. These applications had end-to-end run time reduction of up to 90%, relative to versions that execute tasks via separate scheduler submissions. Furthermore, we also developed an astronomy application named the AstroPortal [7, 8, 80, 81, 82], which has the analysis codes (i.e. image correction, pixel shifting, stacking) and application logic (i.e. mapping between sky coordinates to files) tightly integrated into Falkon. We have deployed the AstroPortal on the TeraGrid [17] distributed infrastructure and it is now online in beta testing by our collaborator's group Alex Szalay at John Hopkins University, with the goal to have it accessible in the future by the broader astronomy community. We have shown the AstroPortal to scale to 100 processors and 100K+ fine grained analysis tasks on the SDSS DR5 dataset [91] with completion times in 2~104 minutes, depending on the data source location [7, 8, 94, 95]; by contrast, a similar scale study which performed a stacking analysis of 41K+ quasars from the same SDSS dataset took 3 months time to perform. [89]

The documents that have been produced thus far and were covered in this section that will provide a good base for the dissertation are:

- **Proposals:**
  - Harnessing Grid Resources to Enable the Dynamic Analysis of Large Astronomy Datasets; NASA GSRP Proposal, Ames Research Center, NASA, 2006 [81]
  - Harnessing Grid Resources to Enable the Dynamic Analysis of Large Astronomy Datasets: Year 1 Status and Year 2 Proposal; NASA GSRP Year 1 Progress Report and Year 2 Proposal, Ames Research Center, NASA, 2007 [82]
  - Falkon: A Proposal for Project Globus Incubation; Globus Incubation Management Project, 2007 [93]
- **Journal/Conference/Workshop Papers and Book Chapters:**
  - AstroPortal: A Science Gateway for Large-scale Astronomy Data Analysis; TeraGrid Conference 2006 [8]
  - Swift: Fast, Reliable, Loosely Coupled Parallel Computation; IEEE Workshop on Scientific Workflows 2007 [13]
  - Falkon: a Fast and Light-weight tasK executiON framework; IEEE/ACM SC07, 2007 [6]
  - Realizing Fast, Scalable and Reliable Scientific Computations in Grid Environments; book chapter in Grid Computing Research Progress, Nova Publisher 2008 [88]
  - Swift: Realizing Fast, Reliable, Large Scale Scientific Computation; under review at Journal of Future Generation Computer Systems 2008 [97]
- **Short Papers:**
  - The Importance of Data Locality in Distributed Computing Applications; NSF Workflow Workshop 2006 [9]
  - A Data Diffusion Approach to Large Scale Scientific Exploration; Microsoft eScience Workshop at RENCI 2007 [79]
- **Posters:**
  - Dynamic Resource Provisioning in Grid Environments; TeraGrid Conference 2007 [11]
  - Harnessing Grid Resources to Enable the Dynamic Analysis of Large Astronomy Datasets; IEEE/ACM SC06, 2006 [7]
- **Technical Reports:**
  - Characterizing Storage Resources Performance in Accessing the SDSS Dataset; Technical Report, University of Chicago, 2005 [94]
  - Characterizing the SDSS DR4 Dataset and the SkyServer Workloads; Technical Report, University of Chicago, 2006 [95]
  - 3DcacheGrid: Dynamic Distributed Data Cache Grid Engine; Technical Report, University of Chicago, 2006 [76]
  - Storage and Compute Resource Management via DYRE, 3DcacheGrid, and CompuStore; Technical Report, University of Chicago, 2006 [77]
  - SkyServer Web Service; Technical Report, University of Chicago, 2007 [80]
  - Dynamic Resource Provisioning in Grid Environments; Technical Report, University of Chicago, 2007 [73]
  - Accelerating Large Scale Scientific Exploration through Data Diffusion; Technical Report, University of Chicago, 2007 [78]
- **Work in Progress:**
  - Enabling Serial Job Execution on the BlueGene Supercomputer with Falkon; Wiki Report 2007 [90]
  - Provisioning EC2 Resources; Wiki Report 2007 [96]

5.2 Future Work

Resource Management (3-Tier Architecture & Alternate Technologies): We plan to evolve the Falkon architecture from the current 2-Tier architecture to a 3-Tier one. We are expecting that this architecture change would allow us to introduce more parallelism and distribution of the currently centralized management component in Falkon, and hence offer higher dispatch and execution rates than Falkon currently supports. We are pursuing this work with the goal to have Falkon run at considerably larger scales, such as those found on the latest IBM BlueGene/P (BG/P) that will be online late 2007 at Argonne National Laboratory. The work in porting Falkon to the BG/P will open new opportunities to applications that traditionally could not execute on the BG/P due to the lack of support of task farms. It will be crucial to test the limits of the 3-Tier architecture from a performance point of view to evaluate the appropriateness of Falkon on the BG/P which can scale to 256K processors. Furthermore, we are also working at simplifying the various components in Falkon, including the communication protocols that are internal to the system. We plan to implement the Executor in C (in addition to the one that is already implemented in Java), and offer a proprietary TCP-based communication protocol (as opposed to the existing Web Services protocol) between the Executors and the Dispatcher. This transition should allow Falkon to achieve higher performance due to the lighter weight communication protocol, and allow the Executor to be deployed on computer architectures that do not support Java, such as the IBM BlueGene. We have already made some progress in this direction and have documented it in a Wiki Report [90]. We plan to submit this work for publication to SC08 (deadline in April 2008), but will consider other venues with earlier deadlines if we have the paper draft earlier.

Resource Management (Provisioning): Some of the dynamic resource provisioning work from Falkon was published in the Falkon SC07 paper [6], others were published in the TG07 poster [11], and an extension to the TG07 poster can be found in a technical report [73]. We plan to extend the provisioning work to support multiple sites within a single provisioner (we currently need multiple provisioners to support multiple sites), and perform a performance study to show the application level benefits and transparency of executing across multiple sites. We plan to submit the dynamic resource provisioning work (both past and the proposed multi-site extension) as self inclusive work for publication at HPDC08 (deadline in January 2008). Furthermore, we plan to extend the provisioner to support other abstract resource allocation interfaces; it currently supports GRAM4, which is sufficient for most deployed Grids. We plan to explore the possibility of using the provisioner to allocate virtual resources via the Workspace Service [84, 85, 86, 87] on both Grids and the Amazon Elastic Compute Cloud (EC2) [83]; an initial write-up of our progress in this direction can be found on he Wiki Report [96]. The EC2 computation resource offers unique features in which the accounting and charging occurs with real money (as opposed to imaginary service units that are granted upon a project's inception). Assuming that Amazon is willing to expand and scale the EC2 service indefinitely as demand rises, it offers a unique opportunity to explore new resource allocation schemes that assume that resources could always be acquired for a certain price (as opposed to the batch-scheduled resource allocation in Grids which implies that jobs must wait in a wait queue if not enough resources are available). We expect to publish this work in PODC08 (deadline in February 2008).

Resource Management (Data Diffusion): We plan to complete the Falkon provider (in Swift) implementation to leverage the data diffusion in Falkon that will allow any application (as opposed to just the AstroPortal) running over Falkon to use the proposed data caching transparently; the goal is to potentially have faster application execution times, as well as better scalability. The data diffusion incorporates data caches in executors and data location-aware task scheduling algorithms in the dispatcher of Falkon. Individual executors manage their own cache content, using local cache eviction policies, and communicate changes to cache content to the dispatcher. The dispatcher then associates with each task sent to an executor information about where to find non-cached data. In the future, we will also analyze and (if analysis results are encouraging) experiment with alternative approaches, such as decentralized indices and centralized management of cache content. We have implemented four well-known cache

eviction policies [15]: *Random*, *FIFO* (First In First Out), *LRU* (Least Recently Used), and *LFU* (Least Frequently Used). Data caching at executors implies the need for data-aware scheduling. We implement four policies: *next-available*, *first-available*, *max-cache-hit*, and *max-compute-util*. We plan to submit an in-depth analysis of the work and results (which are currently in a technical report [78]) to SC08 (deadline in April 2008). We also plan to submit an all inclusive journal paper covering Falkon including the resource provisioning and data diffusion to the Journal of Grid Computing (deadline is open ended).

Applications: With the integration of Falkon into the Karajan [19, 13] workflow engine (which is used by the Swift parallel programming system [13]), Swift applications can use Falkon without modification. Leveraging the wide range of scientific applications that are already implemented and configured to run over Swift, we plan to investigate the following applications in depth to show the effectiveness and flexibility of the AMDASK model and Falkon in practice. The applications span various domains (astronomy [7, 8, 12], astro-physics [40], medicine [32], chemistry [38], economics [39]), covering a variety of different datasets, workloads, and analysis codes. Besides the existing publications of the various applications, we expect to publish the new results in some of the papers already mentioned on the data diffusion and resource provisioning. Furthermore, we expect to publish an updated and extended version of the AstroPortal paper [8] to the eScience Conference (deadline in July 2008).

Performance Evaluation: We need to perform extensive performance evaluations on the effects of data diffusion, the various caching policies, various workloads, and data access patterns. Some of the metrics we will measure are: 1) application execution time, 2) application speedups, 3) task throughput, 4) scalability, 5) queue wait time, 6) data caching: cache hits vs. cache misses, 7) communication overhead, 8) scheduling overheads, 9) data management overheads, 10) dynamic resource provisioning latency, 11) resource efficiency, and 12) resource wastage. The results from the performance evaluation will be integrated throughout the various new papers discussed in this section.

New Science: We expect to work with the astronomy community at large to get the AstroPortal into production so it can be used to advance the astronomy domain. Our contacts with the astronomy community are 1) Alex Szalay from the Department of Physics and Astronomy at Johns Hopkins University, 2) Jerry C. Yan from the NASA Ames Research Center, and 3) the US National Virtual Observatory (NVO) at http://sandbox.us-vo.org/grid.cfm. If significant new science is achieved as a direct result of our work, we expect to publish these new science results in well known astronomy journals with Alex Szalay as the lead author. We are also hoping to help other groups that are currently working with the Swift system to scale their applications to larger scales and faster execution times, and document the new findings for each application, and incorporate interesting findings and results within the various papers mentioned thus far.

Simulations: We will validate the AMDASK model through simulations, to verify the both the efficient and scalability given a wide range of simulation parameters (i.e. number of storage and computational resources, communication costs, management overhead, and workloads – including inter-arrival rates, query complexity, and data locality). We will implement the model in a discrete event simulation that will allow us to investigate a wider parameter space than we could in a real world implementation. The simulations will specifically attempt to model a grid computing environment comprising of computational resources, storage resources, batch schedulers, various communication technologies, various types of applications, and workload models. We will perform careful and extensive empirical performance evaluations in order to create correct and accurate input models to the simulator; the input models include 1) Communication costs, 2) Data management costs, 3) Task scheduling costs, 4) Storage access costs, and 5) Workload models. We expect to be able to scale simulations to more computational and storage resources than we could achieve in a real deployed system due to the availability of resources. Furthermore, assuming the input models to be correct, we should be able to accurately measure the end-to-end performance of various applications using a wide range of strategies for the various resource management components. We expect to publish the abstract task model and its validation via simulations at SC08 (deadline in April 2008), or at SigMetrics08 (deadline in November 2008).

Falkon Visibility and Support: We expect to make the Falkon framework available to the larger Grid community through the Globus Incubator Project initiative [92], and support the Falkon code-base through mailing lists, regular code maintenance, updates. [93]

In summary, the paper topics we plan to submit for publication over the course of the next year are, as we discussed them in Section 5.2:

- *Extending Provisioning to use Virtual Resources in EC2*; PODC08 (February 2008)
- *Data Diffusion*; SC08 (April 2008)
- *Dynamic Resource Provisioning*; SC08 (April 2008)
- *Enabling Serial Job Execution on the BlueGene Supercomputer*; SC08 (April 2008)
- *Falkon v2.0*; Journal of Grid Computing (summer 2008)
- *AstroPortal v2.0: Leveraging Falkon and Data Diffusion*; eScience08 (July 2008)
- *AMDASK Model and Validation* ; SigMetrics08 (November 2008)
- *New Science Achieved with AstroPortal*; an astronomy journal (open ended)

5.3 Timeline of Future Work

Figure 4 outlines the detailed plan of work for the remainder of the five quarters of work, assuming that I can stay on schedule. The organization of the Gantt chart is similar to that found in the future work in section 5.2, in which we break down the work into resource management (3-tier architecture, alternative technologies, provisioning extensions, data diffusion), application case studies (testing and new science), performance evaluation, model validation & simulations, Falkon visibility & support, and the dissertation. There is much work to be done, and to ensure that we can accomplish everything we are proposing, I will have to be focused on the problems outlined in this proposal, and to not have any major deviations from the schedule due to unforeseen things, such as problems with the implementations, communication with application domain experts, or deviations on tangents that are not directly related to the future work mentioned in Section 5.2.

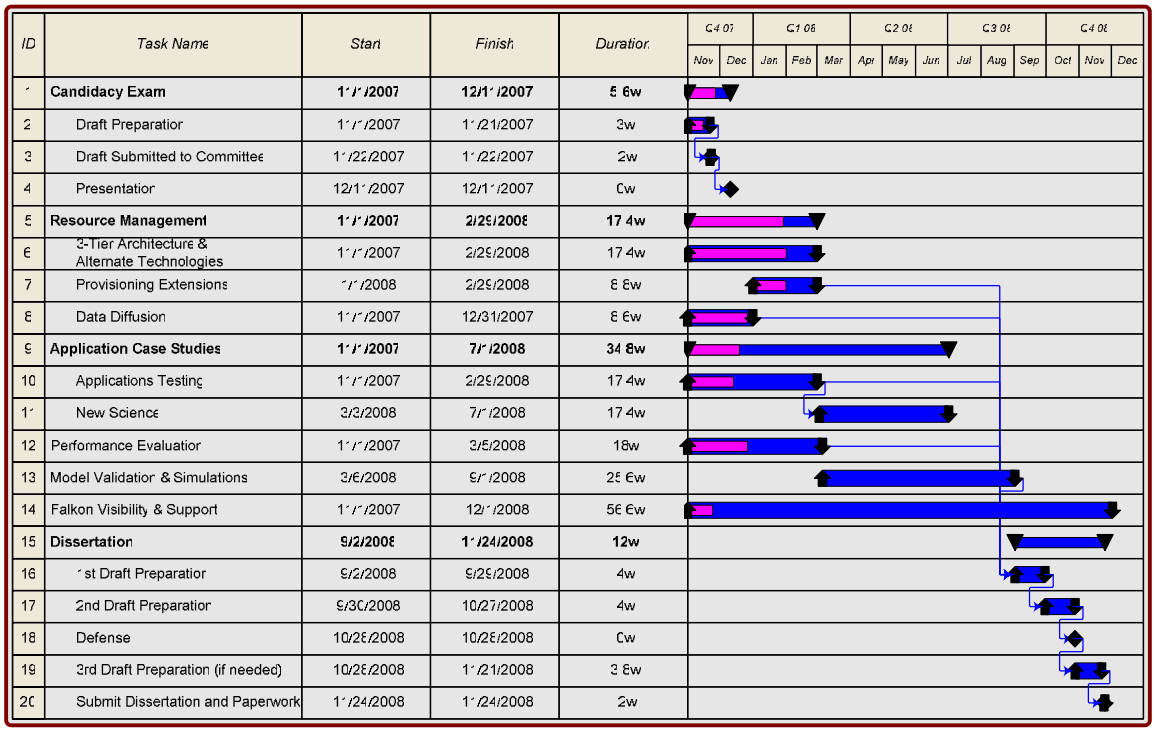


Figure 4: Gantt chart outlining the tentative future work schedule (the purple bars overlaid over the blue bars indicate the approximate amount of work completed for the corresponding row)

The major milestones are the Candidacy Exam which will hopefully take place at the beginning of December 2007, with a draft of the dissertation to follow ten months later and finally with a final defense of the dissertation work in the fall quarter of 2008.

Table 1: Major milestones and dates

Date	Description
11/21/07	Candidacy Paper Draft Sent to Committee
12/12/07	Candidacy Exam
09/29/08	Dissertation Draft
10/28/08	Dissertation Defense

6 Conclusions

We see the dynamic analysis of large datasets to be important due to the ever growing datasets that need to be accessed by larger and larger communities. Attempting to address the storage and computational problems separately essentially forcing much data movement between computational and storage resources will not scale to tomorrow's peta-scale datasets and will likely yield significant underutilization of the raw resources. We defined the abstract model for data-centric task farms, AMDASK, in order to address the integration of the storage and computational issues found in a class of applications which can be decomposed down into many independent computational tasks which need to work on large datasets. We plan to validate the abstract model via discrete event simulations, and to provide a reference implementation to show the flexibility and effectiveness of it on real world applications and datasets.

There are various fundamental research questions we hope to address through our work presented in this proposal. They center on two main areas, data and compute resource management, and how they relate to particular workloads of data analysis on large datasets.

Compute resource management: Dynamic resource provisioning architectures and implementations must be carefully designed in order to offer the right abstraction while at the same time offer practical and measurable advantages over static resource provisioning; dynamic resource provisioning can lead to significant savings in end-to-end time to completion of application runtimes. Another important issue is the scheduling of computational tasks close to the data. Essentially, we need to investigate various strategies for workload management in which we can quantify the cost of moving the work vs. moving the data. Finally, we emphasize the need for high throughput task dispatch in order to support workloads involving many small tasks on medium to large scale Grids.

Data resource management: We believe that data management architectures is important to ensure that the data management implementations scale to the required dataset sizes in the number of files, objects, and dataset disk space usage while at the same time, ensuring that data element information can be retrieved fast and efficiently. We plan to investigate the data placement and caching strategies to identify their appropriateness for workloads, datasets, data locality, and access patterns found in the interactive analysis of large datasets.

Finally, we expect to explore several applications from various domains, such as astronomy, astro-physics, medicine, chemistry, and economics in order to show off the flexibility and effectiveness of the AMDASK model and its implementation (Falkon) on real world applications.

7 References

[1] O. Tatebe, N. Soda, Y. Morita, S. Matsuoka, S. Sekiguchi. "Gfarm v2: A Grid file system that supports high-performance distributed and parallel data computing", Computing in High Energy and Nuclear Physics (CHEP04), 2004.

[2] W. Xiaohui, W.W. Li, O. Tatebe, X. Gaochao, H. Liang, J. Jiubin. "Implementing data aware scheduling in Gfarm using LSF scheduler plugin mechanism", International Conference on Grid Computing and Applications (GCA'05), pp.3-10, 2005.

[3] M. Ernst, P. Fuhrmann, M. Gasthuber, T. Mkrtchyan, C. Waldman. "dCache, a distributed data storage caching system," Computing in High Energy and Nuclear Physics (CHEP01), 2001.

[4] P. Fuhrmann. "dCache, the commodity cache," Twelfth NASA Goddard and Twenty First IEEE Conference on Mass Storage Systems and Technologies 2004.

[5] S. Ghemawat, H. Gobioff, S.T. Leung. "The Google file system," 19th ACM SOSP, 2003.

[6] I. Raicu, Y. Zhao, C. Dumitrescu, I. Foster, M. Wilde. "Falkon: a Fast and Light-weight tasK executiON framework", IEEE/ACM International Conference for High Performance Computing, Networking, Storage, and Analysis (SC07), 2007.

[7] I. Raicu, I. Foster, A. Szalay. "Harnessing Grid Resources to Enable the Dynamic Analysis of Large Astronomy Datasets", IEEE/ACM International Conference for High Performance Computing, Networking, Storage, and Analysis (SC06), 2006.

[8] I. Raicu, I. Foster, A. Szalay, G. Turcu. "AstroPortal: A Science Gateway for Large-scale Astronomy Data Analysis", TeraGrid Conference 2006.

[9] A. Szalay, J. Bunn, J. Gray, I. Foster, I. Raicu. "The Importance of Data Locality in Distributed Computing Applications", NSF Workflow Workshop 2006.

[10] F. Schmuck and R. Haskin, "GPFS: A Shared-Disk File System for Large Computing Clusters," Conference on File and Storage Technologies (FAST), 2002.

[11] I. Raicu, C. Dumitrescu, I. Foster. "Dynamic Resource Provisioning in Grid Environments", TeraGrid Conference 2007.

[12] J.C. Jacob, et al. "The Montage Architecture for Grid-Enabled Science Processing of Large, Distributed Datasets." Earth Science Technology Conference 2004.

[13] Y. Zhao, M. Hategan, B. Clifford, I. Foster, G. von Laszewski, I. Raicu, T. Stef-Praun, M. Wilde. "Swift: Fast, Reliable, Loosely Coupled Parallel Computation", IEEE Workshop on Scientific Workflows 2007.

[14] R. Hasan, Z. Anwar, W. Yurcik, L. Brumbaugh, R. Campbell. "A Survey of Peer-to-Peer Storage Techniques for Distributed File Systems", International Conference on Information Technology: Coding and Computing (ITCC'05) - Volume II - Volume 02, 2005.

[15] S. Podlipnig, L. Böszörmenyi. "A survey of Web cache replacement strategies", ACM Computing Surveys (CSUR), Volume 35 , Issue 4, Pages: 374 – 398, 2003.

[16] R. Lancellotti, M. Colajanni, B. Ciciani, "A Scalable Architecture for Cooperative Web Caching", Workshop in Web Engineering, Networking 2002, 2002.

[17] C. Catlett, et al., "TeraGrid: Analysis of Organization, System Architecture, and Middleware Enabling New Types of Applications," HPC 2006.

[18] D. Thain, T. Tannenbaum, and M. Livny, "Distributed Computing in Practice: The Condor Experience" Concurrency and Computation: Practice and Experience, Vol. 17, No. 2-4, pages 323-356, 2005.

[19] Swift Workflow System: www.ci.uchicago.edu/swift

[20] I. Foster, J. Voeckler, M. Wilde, Y. Zhao. "Chimera: A Virtual Data System for Representing, Querying, and Automating Data Derivation", SSDBM 2002.

[21] J.-P Goux, S. Kulkarni, J.T. Linderoth, and M.E. Yoder, "An Enabling Framework for Master-Worker Applications on the Computational Grid," IEEE International Symposium on High Performance Distributed Computing, 2000.

[22] I. Foster, C. Kesselman, S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations", International Journal of Supercomputer Applications, 15 (3). 200-222, 2001.

[23] J. Frey, T. Tannenbaum, I. Foster, M. Frey, S. Tuecke, "Condor-G: A Computation Management Agent for Multi-Institutional Grids," Cluster Computing, vol. 5, pp. 237-246, 2002.

[24] E. Walker, J.P. Gardner, V. Litvin, E.L. Turner, "Creating Personal Adaptive Clusters for Managing Scientific Tasks in a Distributed Computing Environment", Workshop on Challenges of Large Applications in Distributed Environments, 2006.

[25] G. Singh, C. Kesselman E. Deelman. "Performance Impact of Resource Provisioning on Workflows", Technical Report, USC ISI, 2006.

[26] G. Mehta, C. Kesselman, E. Deelman. "Dynamic Deployment of VO-specific Schedulers on Managed Resources", Technical Report, USC ISI, 2006.

[27] E. Robinson, D.J. DeWitt. "Turning Cluster Management into Data Management: A System Overview", Conference on Innovative Data Systems Research, 2007.

[28] B. Bode, D.M. Halstead, R. Kendall, Z. Lei, W. Hall, D. Jackson. "The Portable Batch Scheduler and the Maui Scheduler on Linux Clusters", Usenix Linux Showcase & Conference, 2000.

[29] S. Zhou. "LSF: Load sharing in large-scale heterogeneous distributed systems," Workshop on Cluster Computing, 1992.

[30] D.P. Anderson. "BOINC: A System for Public-Resource Computing and Storage." IEEE/ACM International Workshop on Grid Computing. November 8, 2004.

[31] D.P. Anderson, E. Korpela, R. Walton. "High-Performance Task Distribution for Volunteer Computing." IEEE International Conference on e-Science and Grid Technologies, 2005.

[32] The Functional Magnetic Resonance Imaging Data Center, http://www.fmridc.org/, 2007.

[33] K. Appleby, S. Fakhouri, L. Fong, G. Goldszmidt, M. Kalantar, S. Krishnakumar, D. Pazel, J. Pershing, and B. Rochwerger, "Oceano - SLA Based Management of a Computing Utility," IFIP/IEEE International Symposium on Integrated Network Management, 2001.

[34] L. Ramakrishnan, L. Grit, A. Iamnitchi, D. Irwin, A. Yumerefendi, J. Chase. "Toward a Doctrine of Containment: Grid Hosting with Adaptive Resource Control," IEEE/ACM International Conference for High Performance Computing, Networking, Storage, and Analysis (SC06), 2006.

[35] J. Bresnahan. "An Architecture for Dynamic Allocation of Compute Cluster Bandwidth", MS Thesis, Department of Computer Science, University of Chicago, 2006.

[36] M. Feller, I. Foster, and S. Martin. "GT4 GRAM: A Functionality and Performance Study", TeraGrid Conference 2007.

[37] The San Diego Supercomputer Center (SDSC) DataStar Log, March 2004 thru March 2005, http://www.cs.huji.ac.il/labs/parallel/workload/l_sdsc_ds/index.html.

[38] V. Nefedova, et al., "Molecular Dynamics (LDRD)", 2007, http://www.ci.uchicago.edu/wiki/bin/view/SWFT/ApplicationStatus.

[39] T. Stef, et al., "Moral Hazard", 2007, http://www.ci.uchicago.edu/wiki/bin/view/SWFT/ApplicationStatus.

[40] ASC / Alliances Center for Astrophysical Thermonuclear Flashes, 2007, http://www.flash.uchicago.edu/website/home/.

[41] R. Buyya and M. Murshed. "GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing", The Journal of Concurrency and Computation: Practice and Experience (CCPE), Volume 14, Issue 13-15, Wiley Press, 2002.

[42] A. Sulistio, C.S. Yeo, and R. Buyya. "A Taxonomy of Computer-based Simulations and its Mapping to Parallel and Distributed Systems Simulation Tools", International Journal of Software: Practice and Experience, Volume 34, Issue 7, Pages: 653-673, Wiley Press, 2004.

[43] GridBus Team, "GridSim: A Grid Simulation Toolkit for Resource Modelling and Application Scheduling for Parallel and Distributed Computing". The GridBus Project, Grid Computing and Distributed Systems (GRIDS) Laboratory, Department of Computer Science and Software Engineering, The University of Melbourne, 2007.

[44] M. Danelutto, "Adaptive Task Farm Implementation Strategies", Euromicro Conference on Parallel, Distributed and Network-based Processing, pp. 416-423, IEEE press, ISBN 0-7695-2083-9, 2004, A Coruna (E), 11-13 February 2004.

[45] E. Heymann, M.A. Senar, E. Luque, and M. Livny, "Adaptive Scheduling for Master-Worker Applications on the Computational Grid", IEEE/ACM International Workshop on Grid Computing (GRID00), 2000.

[46] H. González-Vélez, "An Adaptive Skeletal Task Farm for Grids", International Euro-Par 2005 Parallel Processing, pp. 401-410, 2005.

[47] H. Casanova, M. Kim, J.S. Plank, J.J. Dongarra, "Adaptive Scheduling for Task Farming with Grid Middleware", International Euro-Par Conference, 1999.

[48] D. Petrou, G.A. Gibson, G.R. Ganger, "Scheduling Speculative Tasks in a Compute Farm", IEEE/ACM International Conference for High Performance Computing, Networking, Storage, and Analysis (SC05), 2005.

[49] F.J.L. Reid, "Task farming on Blue Gene", EEPC, Edinburgh University, 2006.

[50] O. Tatebe, N. Soda, Y. Morita, S. Matsuoka, S. Sekiguchi, "Gfarm v2: A Grid file system that supports high-performance distributed and parallel data computing", Computing in High Energy and Nuclear Physics (CHEP04), 2004.

[51] O. Tatebe, Y. Morita, S. Matsuoka, N. Soda, S. Sekiguchi, "Grid Datafarm Architecture for Petascale Data Intensive Computing", IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2002), pp.102-110, 2002

[52] N. Yamamoto, O. Tatebe, S. Sekiguchi, "Parallel and Distributed Astronomical Data Analysis on Grid Datafarm", IEEE/ACM International Workshop on Grid Computing (Grid 2004), pp.461-466, 2004.

[53] W.H. Bell, D.G. Cameron, L. Capozza, A.P. Millar, K. Stockinger, and F. Zini. "OptorSim - A Grid Simulator for Studying Dynamic Data Replication Strategies", International Journal of High Performance Computing Applications, 17(4), 2003.

[54] C. Nicholson, D.G. Cameron, A.T. Doyle, A.P. Millar, K. Stockinger, "Dynamic Data Replication in LCG 2008", UK e-Science All Hands Conference 2006.

[55] A.T. Doyle, C. Nicholson. "Grid Data Management: Simulations of LCG 2008", Computing in High Energy and Nuclear Physics (CHEP06), 2006.

[56] A. Sulistio, U. Cibej, B. Robic and R. Buyya. "A Tool for Modeling and Simulation of Data Grids with Integration of Data Storage, Replication and Analysis", Technical Report, GRIDS-TR-2005-13, Grid Computing and Distributed Systems Laboratory, University of Melbourne, Australia, 2005.

[57] N. Muthuvelu, J. Liu, N.L. Soe, S. Venugopal, A. Sulistio and R. Buyya, "A Dynamic Job Grouping-Based Scheduling for Deploying Applications with Fine-Grained Tasks on Global Grids", Australasian Workshop on Grid Computing and e-Research (AusGrid 2005), 2005.

[58] K. Ranganathan and I. Foster, "Simulation Studies of Computation and Data Scheduling Algorithms for Data Grids", Journal of Grid Computing, V1(1) 2003.

[59] A.L. Chervenak, N. Palavalli, S. Bharathi, C. Kesselman, R. Schwartzkopf, "The Replica Location Service", International Symposium on High Performance Distributed Computing Conference (HPDC-13), June 2004.

[60] A. Chervenak, R. obert Schuler. "The Data Replication Service", Technical Report, USC ISI, 2006.

[61] M. Branco, "DonQuijote - Data Management for the ATLAS Automatic Production System", Computing in High Energy and Nuclear Physics (CHEP04), 2004.

[62] D. Olson and J. Perl, "Grid Service Requirements for Interactive Analysis", PPDG CS11 Report, September 2002.

[63] S. Langella, S. Hastings, S. Oster, T. Kurc, U. Catalyurek, J. Saltz. "A Distributed Data Management Middleware for Data-Driven Application Systems" IEEE International Conference on Cluster Computing, 2004.

[64] A. Chervenak, E. Deelman, C. Kesselman, B. Allcock, I. Foster, V. Nefedova, J. Lee, A. Sim, A. Shoshani, B. Drach, D. Williams, D. Middleton. "High-Performance Remote Access to Climate Simulation Data: A Challenge Problem for Data Grid Technologies", Parallel Computing, Special issue: High performance computing with geographical data, Volume 29 , Issue 10 , pp 1335 – 1356, 2003.

[65] M. Beynon, T.M. Kurc, U.V. Catalyurek, C. Chang, A. Sussman, J.H. Saltz. "Distributed Processing of Very Large Datasets with DataCutter", Parallel Computing, Vol. 27, No. 11, pp. 1457-1478, 2001.

[66] D.L. Adams, K. Harrison, C.L. Tan. "DIAL: Distributed Interactive Analysis of Large Datasets", Conference for Computing in High Energy and Nuclear Physics (CHEP 06), 2006.

[67] D.T. Liu, M.J. Franklin. "The Design of GridDB: A Data-Centric Overlay for the Scientific Grid", VLDB04, pp. 600-611, 2004.

[68] F. Chang, J. Dean, S. Ghemawat, W.C. Hsieh, D.A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R.E. Gruber. "Bigtable: A Distributed Storage System for Structured Data", Symposium on Operating System Design and Implementation (OSDI'06), 2006.

[69] R. Pike, S. Dorward, R. Griesemer, S. Quinlan. "Interpreting the Data: Parallel Analysis with Sawzall", Scientific Programming Journal, Special Issue on Grids and Worldwide Computing Programming Models and Infrastructure 13:4, pp. 227-298, 2005.

[70] J. Dean and S. Ghemawat. "MapReduce: Simplified Data Processing on Large Clusters", Symposium on Operating System Design and Implementation (OSDI'04), 2004.

[71] A. Iosup, C. Dumitrescu, D.H.J. Epema, H. Li, L. Wolters, "How are Real Grids Used? The Analysis of Four Grid Traces and Its Implications", IEEE/ACM International Conference on Grid Computing (Grid), 2006.

[72] NIST/SEMATECH e-Handbook of Statistical Methods, http://www.itl.nist.gov/div898/handbook/, 2007.

[73] I. Raicu, Y. Zhao, C. Dumitrescu, I. Foster, M. Wilde. "Dynamic Resource Provisioning in Grid Environments", Technical Report, University of Chicago, 2007, http://people.cs.uchicago.edu/~iraicu/research/reports/up/DRP_v01.pdf.

[74] TeraPort Cluster, http://teraport.uchicago.edu/, 2007.

[75] Tier3 Cluster, http://twiki.mwt2.org/bin/view/UCTier3/WebHome, 2007.

[76] I. Raicu, I. Foster, A. Szalay. "3DcacheGrid: Dynamic Distributed Data Cache Grid Engine", Technical Report, University of Chicago, 2006, http://people.cs.uchicago.edu/~iraicu/research/reports/up/HPC_SC_2006_v09.pdf.

[77] I. Raicu, I. Foster. "Storage and Compute Resource Management via DYRE, 3DcacheGrid, and CompuStore", Technical Report, University of Chicago, 2006, http://people.cs.uchicago.edu/~iraicu/research/reports/up/Storage_Compute_RM_Performance_06.pdf.

[78] I. Raicu, Y. Zhao, I. Foster, A. Szalay. "Accelerating Large Scale Scientific Exploration through Data Diffusion," Technical Report, University of Chicago, 2007, http://people.cs.uchicago.edu/~iraicu/research/reports/up/falkon_data-diffusion_v04.pdf.

[79] I. Raicu, Y. Zhao, I. Foster, A. Szalay. "A Data Diffusion Approach to Large Scale Scientific Exploration," Microsoft eScience Workshop at RENCI 2007.

[80] I. Raicu, I. Foster. "SkyServer Web Service", Technical Report, University of Chicago, 2007, http://people.cs.uchicago.edu/~iraicu/research/reports/up/SkyServerWS_06.pdf.

[81] I. Raicu, I. Foster. "Harnessing Grid Resources to Enable the Dynamic Analysis of Large Astronomy Datasets", NASA GSRP Proposal, Ames Research Center, NASA, February 2006 -- Award funded 10/1/06 - 9/30/07, http://people.cs.uchicago.edu/~iraicu/research/reports/up/NASA_proposal06.pdf.

[82] I. Raicu, I. Foster. "Harnessing Grid Resources to Enable the Dynamic Analysis of Large Astronomy Datasets: Year 1 Status and Year 2 Proposal", NASA GSRP Year 1 Progress Report and Year 2 Proposal, Ames Research Center, NASA, February 2007 -- Award funded 10/1/07 - 9/30/08, http://people.cs.uchicago.edu/~iraicu/research/reports/up/NASA_proposal07.pdf.

[83] Amazon, "Amazon Elastic Compute Cloud (Amazon EC2)", http://www.amazon.com/gp/browse.html?node=201590011, 2007.

[84] B. Sotomayor, K. Keahey, I. Foster, T. Freeman. "Enabling Cost-Effective Resource Leases with Virtual Machines," HPDC Hot Topics, 2007.

[85] K. Keahey, T. Freeman, J. Lauret, D. Olson. "Virtual Workspaces for Scientific Applications," SciDAC 2007.

[86] R. Bradshaw, N. Desai, T. Freeman, K. Keahey. "A Scalable Approach To Deploying And Managing Appliances," TeraGrid Conference 2007.

[87] B. Sotomayor, "A Resource Management Model for VM-Based Virtual Workspaces", Masters paper, University of Chicago, 2007.

[88] Y. Zhao, I. Raicu, I. Foster, M. Hategan, V. Nefedova, M. Wilde. "Realizing Fast, Scalable and Reliable Scientific Computations in Grid Environments", to appear as a book chapter in Grid Computing Research Progress, Nova Publisher 2008.

[89] R.L. White, D.J. Helfand, R.H. Becker, E. Glikman, and W. Vries. "Signals from the Noise: Image Stacking for Quasars in the FIRST Survey", The Astrophysical Journal, Volume 654, pages 99 – 114, 2007.

[90] I. Raicu, I. Foster, Z. Zhang. "Enabling Serial Job Execution on the BlueGene Supercomputer with Falkon," work in progress, http://www.ci.uchicago.edu/wiki/bin/view/VDS/DslCS/Falkon_BG.

[91] SDSS: Sloan Digital Sky Survey, http://www.sdss.org/.

[92] Globus Incubation Management Project, http://dev.globus.org/wiki/Incubator.

[93] I. Raicu, Y. Zhao, C. Dumitrescu, I. Foster and M. Wilde. "Falkon: A Proposal for Project Globus Incubation", Globus Incubation Management Project, 2007, http://people.cs.uchicago.edu/~iraicu/research/reports/up/Falkon-GlobusIncubatorProposal_v3.pdf.

[94] I. Raicu, , I. Foster. "Characterizing Storage Resources Performance in Accessing the SDSS Dataset," Technical Report, University of Chicago, 2005, http://people.cs.uchicago.edu/~iraicu/research/reports/up/astro_portal_report_v1.5.pdf

[95] I. Raicu, , I. Foster. “Characterizing the SDSS DR4 Dataset and the SkyServer Workloads,” Technical Report, University of Chicago, 2006, http://people.cs.uchicago.edu/~iraicu/research/reports/up/SkyServer_characterization_2006.pdf

[96] I. Raicu, C. Dumitrescu, I. Foster. “Provisioning EC2 Resources,” work in progress, http://www.ci.uchicago.edu/wiki/bin/view/VDS/DslCS/Falkon_EC2.

[97] Y. Zhao, I. Raicu, M. Hategan, M. Wilde, I. Foster. “Swift: Realizing Fast, Reliable, Large Scale Scientific Computation”, under review at Journal of Future Generation Computer Systems.