Resource Management in Large-Scale Distributed Systems

Ioan Raicu, Alok Choudhary
Electrical Engineering and Computer Science, Northwestern University
iraicu@eecs.northwestern.edu, *choudhar@eecs.northwestern.edu*

Summary

A famous quote from Ian Foster – "*the advent of computation can be compared, in terms of the breadth and depth of its impact on research and scholarship, to the invention of writing and the development of modern mathematics*" – is central to the growing importance of computational science to the many branches of science, and how interdisciplinary research can bridge the gap among them. Computational science has already begun to change how science is done, enabling scientific breakthroughs through new kinds of experiments that would have been impossible only a decade ago. Today's science is generating datasets that are increasing exponentially in both complexity and volume, making their analysis, archival, and sharing one of the grand challenges of the 21st century. Seymour Cray once said – "*a supercomputer is a device for turning compute-bound problems into I/O-bound problems*" – which drills at the fundamental shift in bottlenecks as supercomputers gain more parallelism in the number of cores at exponential rates, the storage infrastructure performance is increasing at a lower rate. This implies that the data management and data flow between the storage and compute resources is becoming the main bottleneck for large-scale data-intensive applications. The support for data intensive computing is critical to advancing modern science as storage systems have experienced an increasing gap between its capacity and its bandwidth by more than 10-fold over the last decade. There is an emerging need for advanced techniques to manipulate, visualize and interpret large datasets. Many domains share these data management challenges, strengthening the potential road impact from a generic solution. The world of high performance computing is heavily focused on compute-intensive applications, where I/O is assumed (often wrongly) to be a minor overhead, and parallel file systems avoid the need for data locality; the focus is on achieving load balance and reducing communication overheads for fine grain tasks. On the other hand, data-intensive computing is heavily focused on disk access overheads, on data distribution and on the scheduling of computations close to the data; models, such as MapReduce assume that inter-task communication is limited and tasks are coarse grained. We focus on the applications that lie at the intersection producing both compute-intensive and data-intensive workloads, which we have classified in prior work as a new paradigm called Many-Task Computing (MTC).

MTC aims to bridge the gap between two traditional and prevalent programming paradigms for large-scale distributed systems, high throughput computing (HTC) and high performance computing (HPC). MTC is reminiscent to HTC, but it differs in the emphasis of using many computing resources over short periods of time to accomplish many computational tasks, where the primary metrics are measured in seconds, as opposed to operations per month. MTC denotes high-performance computations comprising multiple distinct activities, coupled via file system operations or message passing. There are many challenges to enable support for MTC across clusters, Grids, and supercomputers, including scalable resource management and storage solutions, as well as having well defined standards on how applications are to interact with the new or improved middleware.

This proposal aims to develop both the theoretical and practical aspects of building efficient and scalable support for both compute-intensive and data-intensive MTC. To achieve this, we envision building a new distributed data-aware execution fabric that will support HPC, MTC, and HTC workloads concurrently and efficiently at scales of at least millions of processors and petabytes of storage, and verify through simulations that such a system could scale to a billion processors. Clients will be able to submit computational jobs into the execution fabric by submitting to any compute node (as opposed to submitting to single point of failure gateway nodes), the fabric will guarantee that jobs will execute at least once, and that it will optimize the data movement in order to maximize processor utilization and minimize data transfer costs. The execution fabric will be elastic in which nodes will be able to join and leave dynamically, and data will be automatically replicated throughout the distributed system for both redundancy and performance. We will employ a variety of semantic for the data access patterns, from full POSIX compliance for generality, to relaxed semantics (e.g. eventual consistency on data modifications, write-once read-many data access patterns) to avoid consistency issues and increase scalability. Achieving this level of scalability and transparency will allow the data-aware execution fabric to revolutionize the types of applications that can be supported at petascale and future exascale levels. It will also positively impact the design of future supercomputers to rely more on a distributed architectures, rather than the current common architecture of having the storage completely separate from the compute systems connected via a high speed network interconnect. Since data locality is most critical at the largest scales, the proposed work can lower the costs of new supercomputers by reducing the need for monolithic parallel file systems as well as expensive and exotic network interconnects, while increasing the reliance on distributed storage systems built on commodity hardware and network interconnects. We believe this shift in large-scale architecture design will lead to improving application performance and scalability for the most demanding data intensive applications as system scales continue to increase according to Moore's Law.

The intellectual merit of the proposed work is extremely high, as it will touch every branch of computer science research that is intertwined with computational science by changing the way computing is performed, facilitating rapid analysis at unprecedented speeds and scalability. The impacted application areas will include all traditionally compute intensive disciplines from medicine, astronomy, bioinformatics, chemistry, aeronautics, analytics, economics, to new emerging computational areas in the humanities, arts, and education which are increasingly dealing with ever growing large datasets. We are qualified to pursue the proposed work as we have an in-depth understanding of the challenges and potential solutions, due to our preliminary pioneer work in supporting MTC applications (Swift, Falkon, data diffusion), as well as decades of experience in supporting HPC applications (MPI, PnetCDF, PVFS). We have access to five of the top eight supercomputers in the world (using the June 2009 Top500 rankings) from all of the big supercomputing vendors (Cray, SGI, IBM, and Sun). These five supercomputers represent the majority of supercomputer architectures and are ideal for research and development of novel data management techniques at the frontier of the largest scales available today. We also have access to the TeraGrid (TG) [43], the largest national cyberinfrastructure for open science research, as well as the Open Science Grid (OSG) [44]. Both of these grids offer unique opportunities to explore data management techniques that not only optimize data movement within tightly coupled systems, but also across geographic distribution that brings an entire new range of issues and challenges. Our proposal of enabling MTC on the largest HPC resources of today and tomorrow is controversial as it forces an outside the box thinking and changing the application landscape of high-end computing dramatically from what it has been over the past several decades. However, the de-facto HPC programming model (e.g. MPI-based) was designed in a time when supercomputers had parallelism in the tens or hundreds of processors, and had the capacity that is less than today's hand-held devices. MPI-based applications are already facing scalability walls at the largest scales due to the orders of magnitude larger parallelism, heterogeneous parallelism brought on by the multi-core era, and a decreasing mean-time-to-failure (MTTF) due to increasing system complexity and size. The proposed MTC paradigm has been defined and built with the scalability of tomorrows systems as a priority and can address many of the HPC shortcomings at extreme scales. MTC also fits well with modern parallel programming languages, which will help accelerate the uptake of new thinking methods that until recently were reserved for the select few that were involved in large scale science on distributed systems. These new emerging parallel programming languages will be critical in harnessing these massively parallel machines by non-experts and to allow maximal impact of the many-core computing era.

The quote -- "*The users should be able to focus their attention on the information content of the data, rather than how to discover, access, and use it.*" -- from the Climate Change Science Program report (2003) underscores the broader impact that new and improved data management techniques can have upon the scientific community. In essence, they claim that one of the main objectives of future research programs should be to enhance the data management infrastructure, which is closely aligned with a NSF report on "Research Challenges in Distributed Computing Systems". We believe that delivering innovative support for data-intensive large-scale applications will be an important step forward to achieving the goals set out by the national agencies. By supporting compute intensive and data intensive MTC, we will capitalize NSF's and DOE's investments in the TeraGrid, some of the largest supercomputers available to open science research, as well as to emerging scientific clouds. The challenges we believe we can overcome will only be magnified with the new generation of supercomputers that are scheduled to come online in 2011 that will boast millions of processors and tens of petaflops of compute power, not to mention the expected exascale systems with hundreds of millions to billions of processors that will be available within a decade. The proposed work has a potential for high impact as it addresses resource management challenges and solutions looking forward over ten years into exascale computing and storage. These advancements will impact scientific discovery and economic development at the national level, and it will strengthen a wide range of research and development activities by enabling efficient access, processing, storage, and sharing of valuable scientific data. We will disseminate our results through open source software and publications in the top conferences and journals (e.g. SC, HPDC, NSDI, JGC, and TPDS). Furthermore, this work will serve as the foundation for numerous undergraduate and graduate courses for general computer science literacy across the university, as well as for advanced topics for computer science majors.

This work proposes novel resource management techniques to allow the many branches of science to make use of the dramatic advancements being made in computing and storage systems. The intersection of computer science and the sciences has the potential to have a profound impact on science, how it is practiced, and the rate that major advancements are achieved. Computational science represents the foundation of a new revolution in science that is just beginning, will re-energize virtually all disciplines over the next decades, and will enable a new era of science-based innovation that could dwarf the last decades of technology-based innovation.

1. Overview

A famous quote from Ian Foster – "*the advent of computation can be compared, in terms of the breadth and depth of its impact on research and scholarship, to the invention of writing and the development of modern mathematics*" – is central to the growing importance of computational science to the many branches of science, and how interdisciplinary research can bridge the gap among them. Computational science has already begun to change how science is done, enabling scientific breakthroughs through new kinds of experiments that would have been impossible only a decade ago. Today's science is generating datasets that are increasing exponentially in both complexity and volume, making their analysis, archival, and sharing one of the grand challenges of the 21st century.

1.1. Defining Many-Task Computing

Many-task computing [11, 19, 20] aims to bridge the gap between two computing paradigms, high throughput computing and high performance computing. Many task computing differs from high throughput computing in the emphasis of using large number of computing resources over short periods of time to accomplish many computational tasks (i.e. including both dependent and independent tasks), where primary metrics are measured in seconds (e.g. FLOPS, tasks/sec, MB/s I/O rates), as opposed to operations (e.g. jobs) per month. Many task computing denotes high-performance computations comprising multiple distinct activities, coupled via file system operations. Tasks may be small or large, uniprocessor or multiprocessor, compute-intensive or data-intensive. The set of tasks may be static or dynamic, homogeneous or heterogeneous, loosely coupled or tightly coupled. The aggregate number of tasks, quantity of computing, and volumes of data may be extremely large. Many task computing includes loosely coupled applications that are generally communication-intensive but not naturally expressed using standard message passing interface commonly found in high performance computing, drawing attention to the many computations that are heterogeneous but not “happily” parallel.

We want to enable the use of large-scale distributed systems for task-parallel applications, which are linked into useful workflows through the looser task-coupling model of passing data via files between dependent tasks. This potentially larger class of task-parallel applications is precluded from leveraging the increasing power of modern parallel systems such as supercomputers (e.g. IBM Blue Gene/L [23] and Blue Gene/P [24]) because the lack of efficient support in those systems for the “scripting” programming model [25]. With advances in e-Science and the growing complexity of scientific analyses, more scientists and researchers rely on various forms of scripting to automate end-to-end application processes involving task coordination, provenance tracking, and bookkeeping. Their approaches are typically based on a model of loosely coupled computation, in which data is exchanged among tasks via files, databases or XML documents, or a combination of these. Vast increases in data volume combined with the growing complexity of data analysis procedures and algorithms have rendered traditional manual processing and exploration unfavorable as compared with modern high performance computing processes automated by scientific workflow systems. [26]

The problem space can be partitioned into four main categories (Figure 1 and Figure 2). 1) At the low end of the spectrum (low number of tasks and small input size), we have tightly coupled Message Passing Interface (MPI) applications (white). 2) As the data size increases, we move into the analytics category, such as data mining and analysis (blue); MapReduce [27, 46] is an example for this category. 3) Keeping data size modest, but increasing the number of tasks moves us into the loosely coupled applications involving many tasks (yellow); Swift/Falkon [28, 29] and Pegasus/DAGMan [30] are examples of this category. 4) Finally, the combination of both many tasks and large datasets moves us into the data-intensive many-task computing category (green); examples of this category are Swift/Falkon and data diffusion [31], Dryad [32], and Sawzall [33].

High performance computing can be considered to be part of the first category (denoted by the white area). High throughput computing [34] can be considered to be a subset of the third category (denoted by the yellow area). Many-Task Computing can be considered as part of categories three and four (denoted by the yellow and green areas). This paper focuses on defining many-task computing, and the challenges that arise as datasets and computing systems are growing exponentially.

Clusters and Grids have been the preferred platform for loosely coupled applications that have been traditionally part of the high throughput computing class of applications, which

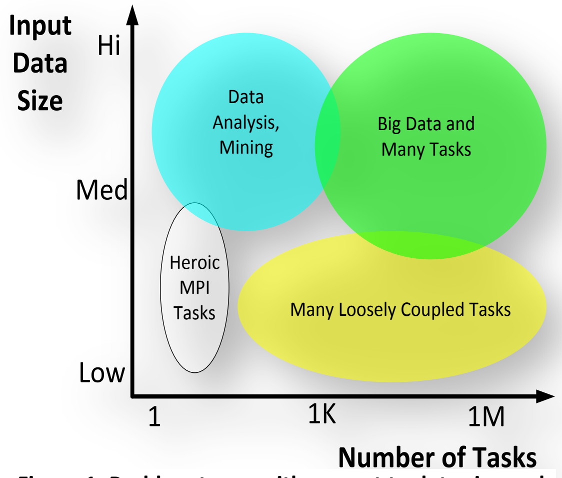


Figure 1: Problem types with respect to data size and number of tasks

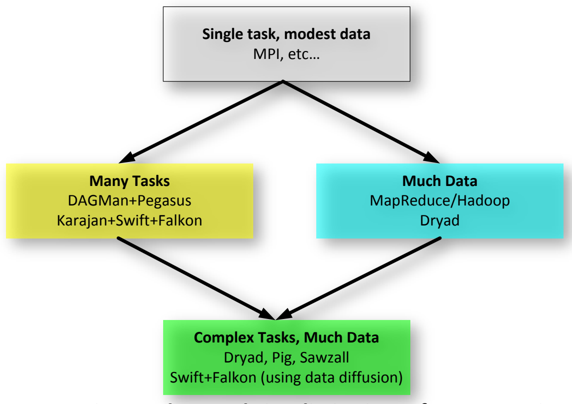


Figure 2: An incomplete and simplistic view of programming models and tools

are managed and executed through workflow systems or parallel programming systems. Various properties of a new emerging applications, such as large number of tasks (i.e. millions or more), relatively short per task execution times (i.e. seconds to minutes long), and data intensive tasks (i.e. tens of MB of I/O per CPU second of compute) have lead to the definition of a new class of applications called Many-Task Computing. MTC emphasizes on using much large numbers of computing resources over short periods of time to accomplish many computational tasks, where the primary metrics are in seconds (e.g., FLOPS, tasks/sec, MB/sec I/O rates), while HTC requires large amounts of computing for long periods of time with the primary metrics being operations per month [34]. MTC applications are composed of many tasks (both independent and dependent tasks) that can be individually scheduled on many different computing resources across multiple administrative boundaries to achieve some larger application goal.

MTC denotes high-performance computations comprising multiple distinct activities, coupled via file system operations or message passing. Tasks may be small or large, uniprocessor or multiprocessor, compute-intensive or data-intensive. The set of tasks may be static or dynamic, homogeneous or heterogeneous, loosely coupled or tightly coupled. The aggregate number of tasks, quantity of computing, and volumes of data may be extremely large. Is MTC really different enough to justify coining a new term? There are certainly other choices we could have used instead, such as multiple program multiple data (MPMD), high throughput computing, workflows, capacity computing, or embarrassingly parallel.

MPMD is a variant of Flynn's original taxonomy [35], used to denote computations in which several programs each operate on different data at the same time. MPMD can be contrasted with Single Program Multiple Data (SPMD), in which multiple instances of the same program each execute on different processors, operating on different data. MPMD lacks the emphasis that a set of tasks can vary dynamically. High throughput computing [34], a term coined by Miron Livny within the Condor project [36], to contrast workloads for which the key metric is not floating-point operations per second (as in high performance computing) but "per month or year." MTC applications are often just as concerned with performance as is the most demanding HPC application; they just don't happen to be SPMD programs. The term "workflow" was first used to denote sequences of tasks in business processes, but the term is sometimes used to denote any computation in which control and data passes from one "task" to another. We find it often used to describe many-task computations (or MPMD, HTC, MTC, etc.), making its use too general. "Embarrassingly parallel computing" is used to denote parallel computations in which each individual (often identical) task can execute without any significant communication with other tasks or with a file system. Some MTC applications will be simple and embarrassingly parallel, but others will be extremely complex and communication-intensive, interacting with other tasks and shared file-systems. The new term MTC is drawing attention to "applications that are communication-intensive but are not naturally expressed in MPI", which are also loosely coupled, and could potentially be composed of many independent tasks. Many of these computations are essentially heterogeneous but not "happily" parallel.

1.2. A Case for Many-Task Computing on Extreme Scale Distributed Systems

We claim that MTC applies to not only traditional HTC environments such as clusters and Grids, assuming appropriate support in the middleware, but also supercomputers [9, 10, 38]. Emerging petascale computing systems, such as IBM's Blue Gene/P [24], incorporate high-speed, low-latency interconnects and other features designed to support tightly coupled parallel computations. Most of the applications run on these computers have a SMPD structure, and are commonly implemented by using MPI to achieve the needed inter-process communication. We believe MTC to be a viable paradigm for supercomputers. As the computing and storage scale increases, the set of problems that must be overcome to make MTC practical (ensuring good efficiency and utilization at large-scale) exacerbate. The challenges include local resource manager scalability and granularity, efficient utilization of the raw hardware, shared file system contention and scalability, reliability at scale, application scalability, and understanding the limitations of the HPC systems in order to identify promising and scientifically valuable MTC applications.

One could ask, why use petascale systems for problems that might work well on terascale systems? We point out that petascale systems are more than just many processors with large peak petaflop ratings. They normally come well balanced, with proprietary, high-speed, and low-latency network interconnects to give tightly-coupled applications good opportunities to scale well at full system scales. Even IBM has proposed in their internal project Kittyhawk [37] that Blue Gene/P can be used to run non-traditional workloads (e.g. HTC). We identify four factors that motivate the support of MTC applications on petascale HPC systems:

1) *The I/O subsystems of petascale systems offer unique capabilities needed by MTC applications.* For example, collective I/O operations [38] could be implemented to use the specialized high-bandwidth and low-latency interconnects. MTC applications could be composed of individual tasks that are themselves parallel programs, many tasks operating on the same input data, and tasks that need considerable communication among them.

Furthermore, the aggregate shared file system performance of a supercomputer can be potentially larger than that found in a distributed infrastructure (i.e., Grid), with data rates in the 10GB+/s range, rather than the more typical 0.1GB/s to 1GB/s range at most Grid sites.

2) *The cost to manage and run on petascale systems like the Blue Gene/P is less than that of conventional clusters or Grids*[37]. For example, a single 13.9 TF Blue Gene/P rack draws 40 kilowatts, for 0.35 GF/watt. Two other systems that get good compute power per watt consumed are the SiCortex with 0.32 GF/watt and the Blue Gene/L with 0.23 GF/watt. In contrast, the average power consumption of the Top500 systems is 0.12 GF/watt [39]. Furthermore, we also argue that it is more cost effective to manage one large system in one physical location, rather than many smaller systems in geographically distributed locations.

3) *Large-scale systems inevitably have utilization issues.* Hence it is desirable to have a community of users who can leverage traditional back-filling strategies to run loosely coupled applications on idle portions of petascale systems.

4) *Perhaps most importantly, some applications are so demanding that only petascale systems have enough compute power to get results in a reasonable timeframe, or to exploit new opportunities in such applications.* With petascale processing capabilities on ordinary applications, it becomes possible to perform vast computations with quick turn-around, thus answering questions in a timeframe that can make a quantitative difference in addressing significant scientific challenges or responding to emergencies.

1.3. The Data Deluge Challenge and the Growing Storage/Compute Gap

A famous quote from Seymour Cray – "*a supercomputer is a device for turning compute-bound problems into I/O-bound problems*" – drills at the fundamental shift in bottlenecks as supercomputers gain more parallelism in the number of cores at exponential rates, yet the storage infrastructure supporting the compute resources is increasing at a lower rate. This implies that the data management and data flow between the storage and compute resources is becoming the main bottleneck for large-scale data-intensive applications. The support for data intensive computing is critical to advancing modern science as storage systems have experienced an increasing gap between its capacity and its bandwidth by more than 10-fold over the last decade. There is an emerging need for advanced techniques to manipulate, visualize and interpret large datasets. [40] Many domains share these data management challenges, strengthening the potential road impact from a generic solution. The world of high performance computing is heavily focused on compute-intensive applications, where I/O is assumed (often wrongly) to be a minor overhead, and parallel file systems avoid the need for data locality; the focus is on achieving load balance and reducing communication overheads for fine grain tasks. On the other hand, data-intensive computing is heavily focused on disk access overheads, on data distribution and on the scheduling of computations close to the data; models, such as Bigtable and MapReduce assume that inter-task communication is limited and tasks are coarse grained. However, many interesting applications do couple modules that are compute-intensive with modules that are data-intensive. This is particularly true for applications in ecology and sustainability that may couple compute-intensive simulations with geographic information systems.

1.4. Background Information -- Prior Work

Our prior work addressed many-task computing at modest scales with the middleware Falkon (a FAst and Light-weight tasK executiON framework) [7, 8] and the parallel programming system Swift [17, 28, 21, 22], as well as other work in resource management in distributed systems and grids [76, 77, 78]. Falkon aims to enable the rapid and efficient execution of many tasks on large compute clusters, and to improve application performance and scalability using novel data management techniques. Falkon combines three techniques to achieve these goals: (1) multi-level scheduling techniques to enable separate treatments of resource provisioning [6] and the dispatch of user tasks to those resources; (2) a streamlined task dispatcher [7] able to achieve order-of-magnitude higher task dispatch rates than conventional schedulers; and (3) data diffusion [13, 31, 15, 16] which performs data caching and uses a data-aware scheduler to leverage the co-located computational and storage resources to minimize the use of shared storage infrastructure. Falkon's integration of multi-level scheduling, streamlined dispatchers, and data diffusion delivers performance not provided by any other system. Falkon has been deployed on various large scale distributed systems, such as the IBM BlueGene/P at Argonne National Laboratory (160K processors) and the NSF TeraGrid. It has also been integrated with other projects, to deliver turn-key solutions on machines such as the BlueGene/P and Sun Constellation where users can have the entire software stack pre-installed and configured at the system level.

We propose to address data intensive many-task computing by re-architecting, parallelizing, and generalizing our resource management techniques we have explored in prior work, with an emphasis on utilizing the largest scale distributed systems of today and tomorrow at petascale and exascale compute levels with millions to billions of processor cores, petabytes of memory, and exabytes of persistent storage. This project proposes to address challenges at the middleware level in large-scale distributed systems, and it especially targets for scalable resource management (including both compute and storage resources), scalable data-aware scheduling, and generic as well as transparent tools to help scientists concentrate on the important scientific application logic, rather than how to implement, execute, or scale their application on distributed systems.

1.5. MTC Applications

We have found many applications that are a better fit for MTC than HTC or HPC. Their characteristics include having a large number of small parallel jobs, a common pattern observed in many scientific applications [28]. They also use files (instead of messages, as in MPI) for intra-processor communication, which tends to make these applications data intensive.

While we can push hundreds or even thousands of such small jobs via GRAM to a traditional local resource manager (e.g. PBS [51], Condor [48], SGE [52]), the achieved utilization of a modest to large resource set will be poor due to high queuing and dispatching overheads, which ultimately results in low job throughput. A common technique to amortize the costs of the local resource management is to "cluster" multiple jobs into a single larger job. Although this lowers the per job overhead, it is best suited when the set of jobs to be executed are homogenous in execution times, or accurate execution time information is available prior to job execution; with heterogeneous job execution times, it is hard to maintain good load balancing of the underlying resource, causing low resource utilization. We claim that "clustering" jobs is not enough, and that the middleware that manages jobs must be streamlined and made as light-weight as possible to allow applications with heterogonous execution times to execute without "clustering" with high efficiency.

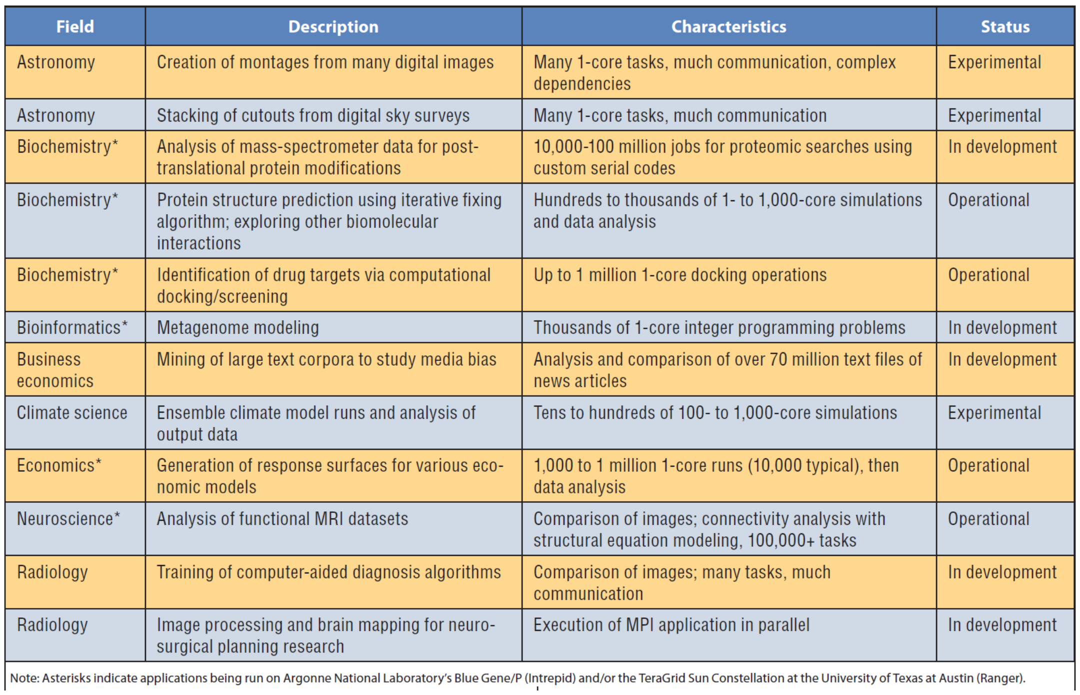
In addition to streamlined task dispatching, scalable data management techniques are also required in order to support MTC applications. MTC applications are often data and/or meta-data intensive, as each job requires at least one input file and one output file, and can sometimes involve many files per job. These data management techniques need to make good utilization of the full network bandwidth of large scale systems, which is a function of the number of nodes and networking technology employed, as opposed to the relatively small number of storage servers that are behind a parallel file system or GridFTP [1] server.

We have identified various applications (as detailed in Table 1 and the rest of this section) from many disciplines that demonstrate characteristics of MTC applications. These applications cover a wide range of domains, from astronomy, physics, astrophysics, pharmaceuticals, bioinformatics, biometrics, neuroscience, medical imaging, chemistry, climate modeling, economics, and data analytics. They often involve many tasks, ranging from tens of thousands to billions of tasks, and have a large variance of task execution times ranging from hundreds of milliseconds to hours. Furthermore, each task is involved in multiple reads and writes to and from files, which can range in size from kilobytes to gigabytes. These characteristics made traditional resource management techniques found in HTC inefficient; also, although some of these applications could be coded as HPC applications, due to the wide variance of the arrival rate of tasks from many users, an HPC implementation would also yield poor utilization. Furthermore, the data intensive nature of these applications can quickly saturate parallel file systems at even modest computing scales.

Astronomy: One of the first applications that motivated much of this work was called the "AstroPortal" [53, 2], which offered a stacking service of astronomy images from the Sloan Digital Sky Survey (SDSS) dataset using grid resources. Astronomical image collections usually cover an area of sky several times (in different wavebands, different times, etc). On the other hand, there are large differences in the sensitivities of different observations: objects detected in one band are often too faint to be seen in another survey. In such cases we still would like to see whether these objects can be detected, even in a statistical fashion. There has been a growing interest to re-project each image to a common set of pixel planes, then stacking images. The stacking improves the signal to noise, and after coadding a large number of images, there will be a detectable signal to measure the average brightness/shape etc of these objects. This application involved the SDSS dataset [41] (currently at 10TB with over 300 million objects, but these datasets could be petabytes in size if we consider multiple surveys in both time and space) [26], many tasks ranging from 10K to millions of tasks, each requiring 100ms to seconds of compute and 100KB to MB of input and output data. [3, 4, 5]

Another related application in astronomy is MONTAGE [54, 55], a national virtual observatory project [56] that stitches tiles of images of the sky from various sky surveys (e.g. SDSS [41], etc) into a photorealistic single image. Execution times per task range in the 100ms to 10s of seconds, and each task involves multiple input images and at least one image output. This application is both compute intensive and data intensive, and has been run as both a HTC (using Pegasus/DAGMan [30], and Condor [36]) and a HPC (using MPI) application, but we found its scalability to be limited when run under HTC or HPC.

Table 1: Sample MTC Applications using the Swift and/or Falkon systems



Field	Description	Characteristics	Status
Astronomy	Creation of montages from many digital images	Many 1-core tasks, much communication, complex dependencies	Experimental
Astronomy	Stacking of cutouts from digital sky surveys	Many 1-core tasks, much communication	Experimental
Biochemistry*	Analysis of mass-spectrometer data for post-translational protein modifications	10,000-100 million jobs for proteomic searches using custom serial codes	In development
Biochemistry*	Protein structure prediction using iterative fixing algorithm; exploring other biomolecular interactions	Hundreds to thousands of 1- to 1,000-core simulations and data analysis	Operational
Biochemistry*	Identification of drug targets via computational docking/screening	Up to 1 million 1-core docking operations	Operational
Bioinformatics*	Metagenome modeling	Thousands of 1-core integer programming problems	In development
Business economics	Mining of large text corpora to study media bias	Analysis and comparison of over 70 million text files of news articles	In development
Climate science	Ensemble climate model runs and analysis of output data	Tens to hundreds of 100- to 1,000-core simulations	Experimental
Economics*	Generation of response surfaces for various economic models	1,000 to 1 million 1-core runs (10,000 typical), then data analysis	Operational
Neuroscience*	Analysis of functional MRI datasets	Comparison of images; connectivity analysis with structural equation modeling, 100,000+ tasks	Operational
Radiology	Training of computer-aided diagnosis algorithms	Comparison of images; many tasks, much communication	In development
Radiology	Image processing and brain mapping for neuro-surgical planning research	Execution of MPI application in parallel	In development

Note: Asterisks indicate applications being run on Argonne National Laboratory's Blue Gene/P (Intrepid) and/or the TeraGrid Sun Constellation at the University of Texas at Austin (Ranger).

Astrophysics: Another application is from astrophysics, which analyzes the Flash turbulence dataset (simulation data) [75] from various perspectives, using volume rendering and vector visualization. The dataset is composed of 32 million files (1000 time steps times 32K files) taking up about 15TB of storage resource, and contains both temporal and spatial locality. In the physics domain, the CMS detector being built to run at CERN's Large Hadron Collider [42] is expected to generate over a petabyte of data per year. Supporting applications that can perform a wide range of analysis of the LHC data will require novel support for data intensive applications.

Economic Modeling: An application from the economic modeling domain that we have investigated as a good MTC candidate is Macro Analysis of Refinery Systems (MARS) [50], which studies economic model sensitivity to various parameters. MARS models the economic and environmental impacts of the consumption of natural gas, the production and use of hydrogen, and coal-to-liquids co-production, and seeks to provide insights into how refineries can become more efficient through the capture of waste energy. Other economic modeling applications perform numerical optimization to determine optimal resource assignment in energy problems. This application is challenging as the parameter space is extremely large, which can produce millions, even billions of individual tasks, each with a relatively short execution time of only seconds long.

Pharmaceutical Domain: In the pharmaceutical domain, there are applications that screen KEGG [57] compounds and drugs against important metabolic protein targets using DOCK6 [49] to simulate the "docking" of small molecules, or ligands, to the "active sites" of large macromolecules of known structure called "receptors". A compound that interacts strongly with a receptor (such as a protein molecule) associated with a disease may inhibit its function and thus act as a beneficial drug. The economic and health benefits of speeding drug development by rapidly screening for promising compounds and eliminating costly dead-ends is significant in terms of both resources and human life. The parameter space is quite large, totaling to more than one billion computations that have a large variance of execution times from seconds to hours, with an average of 10 minutes. The entire parameter space would require over 22,600 CPU years, or over 50 days on a 160K processor Blue Gene/P supercomputer [24]. This application is challenging as there many tasks, each task has a wide range of execution times with little to no prior knowledge about its execution time, and involves significant I/O for each computation as the compounds are typically stored in a database (i.e. 10s to 100s of MB large) and must be read completely per computation.

Chemistry: Another application in the same domain is OOPS [58], which aims to predict protein structure and recognize docking partners. In chemistry, specifically in molecular dynamics, we have an application MolDyn whose goal is to calculate the solvation free energy of ligands and protein-ligand binding energy, with structures obtained from the NIST Chemistry WebBook database [59]. Solvation free energy is an important quantity in Computational Chemistry with a variety of applications, especially in drug discovery and design. These applications have similar characteristics as the DOCK application previously discussed.

Bioinformatics: In bioinformatics, Basic Local Alignment Search Tool (BLAST), is a family of tools for comparing primary biological sequence information (e.g. amino-acid sequences of proteins, nucleotides of DNA sequences). A BLAST search enables one to compare a query sequence with a library or database of sequences, and identify library sequences that resemble the query sequence above a certain threshold. [60]. Although the BLAST codes have been implemented in both HTC and HPC, they are often both data and compute intensive, requiring multi-GB databases to be read for each comparison (or kept in memory if possible), and each comparison can be done within minutes on a modern processor-core. MTC and its support for data intensive applications are critical in scaling BLAST on large scale resources with thousands to hundreds of thousands of processors.

Neuroscience Domain: In the neuroscience domain, we have the Computational Neuroscience Applications Research Infrastructure (CNARI), which aims to manage neuroscience tools and the heterogeneous compute resources on which they can enable large-scale computational projects in the neuroscience community. The analysis includes the aphasia study, structural equation modeling, and general use of R for various data analysis. [61] The application workloads involve many tasks, relatively short on the order of seconds, and each task containing many small input and output files making the application meta-data intensive at large scale.

Cognitive Neuroscience: The fMRI application is a workflow from the cognitive neuroscience domain with a four-step pipeline, which includes Automated Image Registration (AIR), Preprocessing and stats from NIH and FMRIB (AFNI and FSL), and Statistical Parametric Mapping (SPM2) [62]. An fMRI Run is a series of brain scans called volumes, with a Volume containing a 3D image of a volumetric slice of a brain image, which is represented by an Image and a Header. Each volume can contain hundreds to thousands of images, and with multiple patients, the number of individual analysis tasks can quickly grow. Task execution times were only seconds long, and the input and output files ranged from kilobytes to megabytes in size. This application could run as an HTC one at small scales, but needs MTC support to scale up.

Data Analytics: Data analytics and data mining is a large field that covers many different applications. Here, we outline several applications that fit MTC well. One example is the analysis of log data from millions computations. Another set of applications are ones commonly found in the MapReduce [63] paradigm, namely “sort” and “word count” [14]. Both of these applications are essential to World Wide Web search engines, and are challenging at medium to large scale due to their data intensive nature. All three applications involve many tasks, many input files (or many disjoint sub-sections of few files), and are data intensive.

Data Mining: Another set of applications that perform data analysis can be classified in the “All-Pairs” class of applications [64]. These applications aim to understand the behavior of a function on two sets, or to learn the covariance of these sets on a standard inner product. Two common applications in All-Pairs are data mining and biometrics. Data mining is the study of extracting meaning from large data sets; one phase of knowledge discovery is reacting to bias or other noise within a set. Different classifiers work better or worse for varying data, and hence it is important to explore many different classifiers in order to be able to determine which classifier is best for that type of noise on a particular distribution of the validation set.

Biometrics: Biometrics aims to identifying humans from measurements of the body (e.g. photos of the face, recordings of the voice, and measurements of body structure). A recognition algorithm may be thought of as a function that accepts two images (e.g. face) as input and outputs a number between zero and one indicating the similarity between the two input images. The application would then compare all images of a database and create a scoring matrix which can later be easily searched to retrieve the most similar images. These All-Pairs applications are extremely challenging as the number of tasks can rapidly grow in the millions and billions, with each task being hundreds of milliseconds to tens of seconds, with multi-megabyte input data per task.

Molecular docking: The DOCK molecular dynamics application is run regularly on Intrepid to simulate the docking of small ligand molecules to large macromolecules (receptors). A compound that interacts strongly with a receptor associated with a disease may inhibit its function and thus prove useful in a beneficial drug. This application is challenging because it involves many tasks, each with a wide range of execution times, and each computation involves significant I/O. Protein description files for docking range from tens to hundreds of megabytes and must be read for each computation. Argonne biochemists use Falkon for molecular docking and surface screening, running at scales of up to 64,000 cores in a single scripted workload.

Uncertainty in economic models: The University of Chicago-Argonne CIM-EARTH project for integrated social, economic, and environmental modeling (www.cim-earth.org) uses Swift on petascale systems to execute parameter sweeps of economic models that forecast energy use and other commodity demands to examine the effects of uncertainty. CIM-EARTH researchers use the parallel scripting paradigm to refine several models for exploring uncertainty through large-scale parallelism. Researchers analyzed thousands of samples from a perturbed input dataset in parallel on Ranger and other parallel systems of thousands of cores each. The model evaluates relative sensitivity to uncertainty (percent from the mean) for consumer and industrial demand for electricity in eight geographical regions. The dark-blue and light-blue envelopes are one and two standard deviations from the mean.

Structural equation modeling: The University of Chicago's Human Neuroscience Laboratory has developed a computational framework for a data-driven approach to structural equation modeling8 (SEM) and has implemented several parallel scripts for modeling functional MRI data within this framework. The Computational Neuroscience Applications Research Infrastructure8 (CNARI, www.cnari.org) uses Swift to execute hundreds of thousands of simultaneous processes running the R data analysis language, consisting of self-contained structural equation models, on Ranger. These self-contained R processing jobs are data objects generated by OpenMx (http://openmx.psyc.virginia.edu), a structural equation modeling package for R that can generate a single model object containing the matrices and algebraic information necessary to estimate the model's parameters. With the CNARI framework, neuroscientists run OpenMx from Swift scripts to conduct exhaustive searches of the model space.

Posttranslational protein modification: The University of Chicago's Ben May Department for Cancer Research is applying petascale parallel scripting to the analysis of posttranslational protein modifications (PTMs), complex changes to proteins that play essential roles in protein function and cellular physiology. The PTMap application takes in raw data files from massspectrometry analysis of biological samples, along with the entire set of sequences of the organism's proteome, and searches them for statistically significant evidence of unidentified PTMs. The tool reads in a mass-spectrometry file—typically 200 megabytes of data in mzXML format— and protein sequences in FASTA format. The analysis of a mass-spectrometry run for a single proteome has abundant opportunities for parallelization at the extreme scale. Researchers want to apply the latest version of PTMap to identify unknown PTMs across a wide range of organisms including E. coli, yeast, cows, mice, and humans.

MPI Ensembles: Finally, another class of applications is managing an ensemble of MPI applications. One example is from the climate modeling domain, which has been studying climate trends and predicting global warming [65], is already implemented as an HPC MPI application. However, the current climate models could be run as ensemble runs (many concurrent MPI applications) to quantify climate model uncertainty. This is challenging in large scale systems such as supercomputers (a typical resource such models would execute on), as the local resource managers (e.g. Cobalt) favor large jobs and have policy against running many jobs at the same time (i.e. where many is more than single digit number of jobs per user).

All these applications pose significant challenges to traditional resource management found in HPC and HTC, from both job management and storage management perspective, and are in critical need of MTC support as the scale of these resources grows.

1.6. Proposed Work

This proposal aims to develop both the theoretical and practical aspects of building efficient and scalable support for both compute-intensive and data-intensive MTC. To achieve this, we envision building a new distributed data-aware execution fabric that scales to at least millions of processors and petabytes of storage, and will support HPC, MTC, and HTC workloads concurrently and efficiently. Clients will be able to submit computational jobs into the execution fabric by submitting to any compute node (as opposed to submitting to single point of failure gateway nodes), the fabric will guarantee that jobs will execute at least once (jobs could be executed more than once under certain conditions such as unusual delays or transient failures), and that it will optimize the data movement in order to maximize processor utilization and minimize data transfer costs. The execution fabric will be elastic in which nodes will be able to join and leave dynamically, and data will be automatically replicated throughout the distributed system for both redundancy and performance.

We will employ a variety of semantic for the data access patterns, from full POSIX compliance for generality, to relaxed semantics (e.g. eventual consistency on data modifications, write-once read-many data access patterns) to avoid consistency issues and increase scalability. Achieving this level of scalability and transparency will allow the data-aware execution fabric to revolutionize the types of applications that can be supported at petascale and future exascale levels. We have to understand what disk storage organizations best match the needs of the applications we envisage. Distributed storage systems provide detailed control on the location of data at file granularity, but entail potentially expensive metadata transactions. Parallel file systems provide limited control of data layout but can significantly reduce, potentially, metadata overheads. While continued support to a model of "one-file-per-process" with POSIX semantics is necessary, it should be possible to bridge the gap between the two models for applications that use I/O libraries such as HDF5 and netCDF.

Support for MTC workloads combines traditional batch oriented HPC workloads with more MTC-like workloads that seeks fast response time on small tasks. With support for data-aware scheduling, coupled with a better handling of the memory hierarchy and I/O, we believe MTC could support interactive HPC (also known as ensemble MPI applications), enabling the execution of thousands of parallel computations each involving many thousands of nodes and years of compute time in minutes of wall-clock time. Efficient support for interactive HPC can significantly alter the use of HPC in industry, as HPC applications that perform advanced simulations, parametric search and uncertainty quantification become part of an interactive design cycle, rather than an overnight validation cycle. One example is from the climate modeling domain, which has been studying climate trends and predicting global warming, is already implemented as an HPC MPI application. However, the current climate models could be run as ensemble runs (many separate concurrent MPI applications) to quantify climate model uncertainty in order to optimize the speedup and efficiency curves of the particular application (e.g. when scalability is poor). This is challenging in large scale systems such as supercomputers, a typical resource such models would execute on, as the local resource managers favor large jobs and have policy against running many jobs at the same time. Even if scheduling policies would not be the bottleneck, many production resource managers (e.g. Cobalt, SGE, PBS, Condor) have high scheduling and resource provisioning overheads which makes executing ensemble MPI applications only tractable if they are long running. This has implications in both the programming model, degree of parallelism, and reliability of application execution as the mean-time-to-failure (MTTF) generally decreases dramatically as applications and physical resources scale increase. All these applications pose significant challenges to traditional resource management found in HPC and HTC, from both job management and storage management perspective, and are in critical need of MTC support as the scale of these resources grows. We envision adding support for traditional HPC workloads to the Falkon middleware, such as support for MPI-based applications. We will support arbitrary resource allocation sizes, in contrast to other production systems that impose a limit on the smallest allocation size (e.g. for the IBM BlueGene/P at Argonne National Laboratory, the lower bound is 256 processors, and allocation sizes have to be multiples of 256). Furthermore, we will support sharing of data across different MPI applications with direct node-to-node communication, in contrast with having to go through some global file system, essentially leveraging all our work on data management from data diffusion.

We believe there is a wide range of applications that can be made possible, or at least be executed more efficiently, by generalizing Falkon and data diffusion, and applying them to the largest scale distributed systems and applications of today and tomorrow. Data locality is most critical at the largest scales, and it can lower the costs of new supercomputers by reducing the need for monolithic parallel file systems as well as expensive and exotic network interconnects, while increasing the reliance on distributed storage systems built on commodity hardware and network interconnects. We believe this shift in large-scale architecture design will lead to improving application performance and scalability for the most demanding data intensive applications as system scales continue to increase according to Moore's Law.

1.7. Technical Merit of Proposed Research

The intellectual merit of the proposed work is extremely high, as it will touch every branch of computer science research that is intertwined with computational science by changing the way computing is performed, facilitating rapid analysis at unprecedented speeds and scalability. The impacted application areas will include all traditionally compute intensive disciplines from medicine, astronomy, bioinformatics, chemistry, aeronautics, analytics, economics, to new emerging computational areas in the humanities, arts, and education which are increasingly dealing with ever growing large datasets. We are qualified to pursue the proposed work as we have an in-depth understanding of the challenges and potential solutions, due to our preliminary pioneer work in supporting MTC applications (Swift, Falkon, data diffusion), as well as decades of experience in supporting HPC applications (MPI, PnetCDF, PVFS). We have access to five of the top eight supercomputers in the world (using the June 2009 Top500 rankings) from all of the big supercomputing vendors

(Cray, SGI, IBM, and Sun). These five supercomputers represent the majority of supercomputer architectures and are ideal for research and development of novel data management techniques at the frontier of the largest scales available today. We also have access to the TeraGrid, the largest national cyberinfrastructure for open science research, as well as the Open Science Grid. Both of these grids offer unique opportunities to explore data management techniques that not only optimize data movement within tightly coupled systems, but also across geographic distribution that brings an entire new range of issues and challenges. Our proposal of enabling MTC on the largest HPC resources of today and tomorrow is controversial as it forces an outside the box thinking and changing the application landscape of high-end computing dramatically from what it has been over the past several decades. However, the de-facto HPC programming model (e.g. MPI-based) was designed in a time when supercomputers had parallelism in the tens or hundreds of processors, and had the capacity that is less than today's hand-held devices. MPI-based applications are already facing scalability walls at the largest scales due to the orders of magnitude larger parallelism, heterogeneous parallelism brought on by the multi-core era, and a decreasing mean-time-to-failure (MTTF) due to increasing system complexity and size. The proposed MTC paradigm has been defined and built with the scalability of tomorrows systems as a priority and can address many of the HPC shortcomings at extreme scales. MTC also fits well with modern parallel programming languages, which will help accelerate the uptake of new thinking methods that until recently were reserved for the select few that were involved in large scale science on distributed systems. These new emerging parallel programming languages will be critical in harnessing these massively parallel machines by non-experts and to allow maximal impact of the many-core computing era.

1.8. Broader Impact

The quote -- “*The users should be able to focus their attention on the information content of the data, rather than how to discover, access, and use it.*” -- from the Climate Change Science Program report (2003) underscores the broader impact that new and improved data management techniques can have upon the scientific community. In essence, they claim that one of the main objectives of future research programs should be to enhance the data management infrastructure, which is closely aligned with a NSF report on “Research Challenges in Distributed Computing Systems”. We believe that delivering innovative support for data-intensive large-scale applications will be an important step forward to achieving the goals set out by the national agencies. By supporting compute intensive and data intensive MTC, we will capitalize NSF’s and DOE's investments in the TeraGrid, some of the largest supercomputers available to open science research, as well as to emerging scientific clouds. The challenges we believe we can overcome will only be magnified with the new generation of supercomputers that are scheduled to come online in 2011 that will boast millions of processors and tens of petaflops of compute power. The proposed work has a potential for high impact as it addresses resource management challenges and solutions looking forward over ten years into exascale computing and storage. These advancements will impact scientific discovery and economic development at the national level, and it will strengthen a wide range of research and development activities by enabling efficient access, processing, storage, and sharing of valuable scientific data. We will disseminate our results through open source software and publications in the top conferences and journals (e.g. SC, HPDC, NSDI, JGC, and TPDS). Furthermore, this work will serve as the foundation for numerous undergraduate and graduate courses for general computer science literacy across the university, as well as for advanced topics for computer science majors.

1.9. Applicability of Proposed Work in other Domains

The ideas (and even implementations) that underpin our work in resource management at the largest scales can be applied to new emerging paradigms, such as Cloud Computing [18]. The Cloud Computing concept surfaced in 2007, although it is not a completely new concept; it has intricate connection to the thirteen-year established Grid Computing paradigm, and other relevant technologies such as utility computing, cluster computing, and distributed systems in general. In building the future Cloud Computing infrastructure, I believe it needs to support on-demand provisioning of “virtual systems” providing the precise capabilities needed by an end-user, something we have been working on for years in the more general context of distributed systems. There is a growing demand to define protocols that allow users and service providers to discover and hand off demands to other providers, to monitor and manage their reservations, and arrange payment. Tools need to be defined and implemented for managing both the underlying resources and the resulting distributed computations. With Cloud Computing infrastructure being a potential significant asset to the scientific community in the future, and the great similarities found between Cloud Computing and more mature computing systems, we believe that our work in large-scale resource management to support compute intensive and data intensive MTC workloads is directly applicable to Cloud Computing in practice.

Furthermore, with the advent of Many-Core Computing, some are predicting that desktop machines will reach thousands of cores within a decade. This increase in parallelism will bring many challenges to end-users, with new programming models and new thinking methods that until recently were reserved for the select few that were involved in large scale science on distributed systems. Having new tools (e.g. parallel programming languages) to harness these massively parallel machines by non-experts will be critical to allow maximal impact of the many-core computing era. New programming models such as MapReduce have made good progress in this direction, however MapReduce by definition is quite constrained, and cannot express a large number of scientific applications. Data-flow driven parallel programming systems (also known as workflow systems in some communities) can express application control flow with direct acyclic graphs (DAG), and are arguably more general than MapReduce. Given the right level of abstraction, these parallel programming systems can provide the same level of ease of use as MapReduce, while being more general and powerful for the scientific community at large.

2. Technical Approach and Tasks

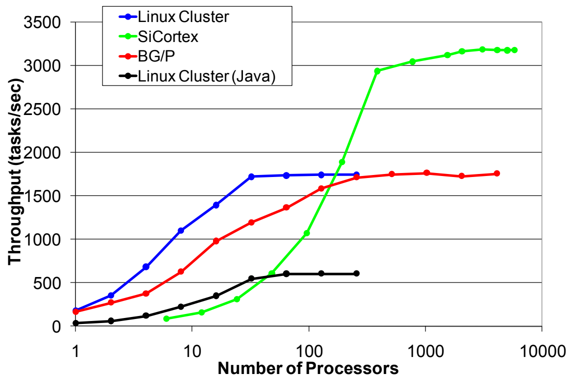
Although the existing middleware (Falkon) and its data management capabilities (data diffusion) have shown to provide significant performance and scalability improvements for a number of scientific applications, they also have several limitations which can hinder wider adoption by the scientific community. These limitations include a centralized data-aware scheduler that scales only to modest levels of thousands of processors, a naïve decentralized scheduler that performs sub-optimal in heterogeneous systems and does not perform data aware scheduling, assumptions on the data access patterns (e.g. write once read many), the requirement that task definitions include input/output files metadata (e.g. file names), the requirement that per task working set fit in local storage, the requirement of a Java virtual machine (which is not typical supported by the largest supercomputers’ compute nodes), and the lack of support for HPC workloads (e.g. MPI-based applications). Furthermore, the Swift parallel programming system currently only scales to tens of thousands of processors, and hundreds of thousands of tasks, due to expensive in-memory state management at a central location. We will discuss in depth the technical solutions we envision can address many of the current shortcomings of the current state-of-the-art parallel programming systems and resource management frameworks.

We will achieve orders of magnitude higher levels of performance by re-architecting the existing centralized Falkon framework to include a distributed queuing system, distributed metadata management, distributed storage systems, and distributed data-aware scheduling. In order to achieve this performance, we will also relax some constraints to ensure our system is able to scale well, such as adopting eventual consistency semantics of the metadata of the storage system, and not guaranteeing ordering of jobs in the distributed queuing system.

2.1. Task 1: Distributed Resource Management

Distributed Queuing System

We will first address the distribution of the resource management framework. Our centralized solution performs quite well (several orders of magnitude better) in comparison with existing production systems, improving throughputs from single digit jobs/sec, to thousands of jobs/sec. However, a centralized solution has two limitations: 1) it has a single point of failure, and 2) it has relatively low scalability as the system is bound to the resources (i.e. processors, memory, network) of a single node to process all jobs. Figure 3 below shows the dispatch rates of Falkon on various systems from a plain Linux cluster of 256 processors, to a 5732 processor SiCortex machine, to a 160K IBM BlueGene/P supercomputer. We see that in all four instances, the throughput saturates early on with only 10s or 100s of processors. Even when running on the full 160K processors on the BlueGene/P, the best throughput we can achieve is 3071 jobs/sec.



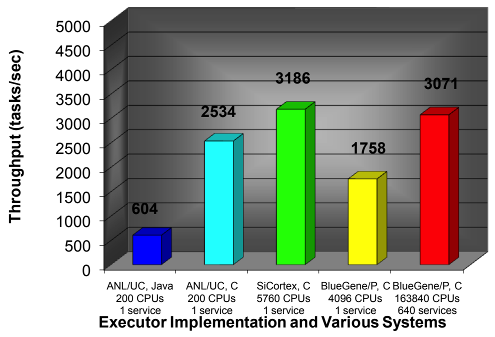
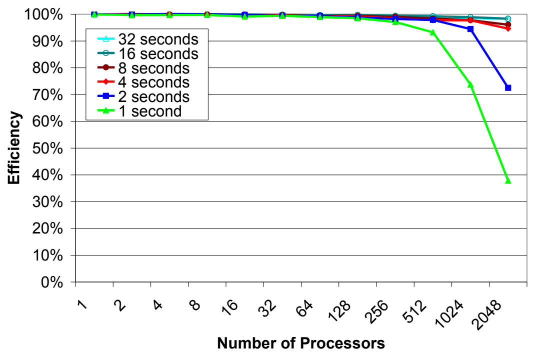


Figure 3: Task dispatch and execution throughput for trivial tasks with no I/O (sleep 0)

The fact that the throughput gets saturated means that at certain work granularity, our system will not operate efficiently. Figure 4 shows the efficiency of various work granularity running on the BlueGene/P at different scales, from 1 to 2048 processors (using a single centralized dispatcher), and up to 160K processors running with the naive distributed dispatchers.



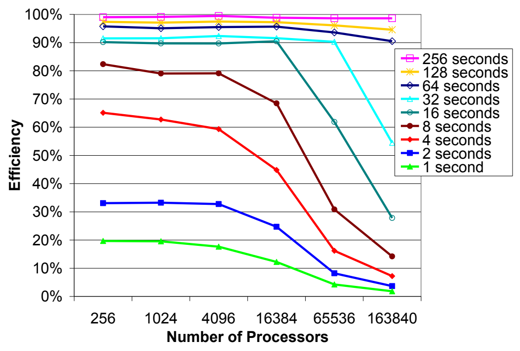


Figure 4: Left: Efficiency graph for the Blue Gene/P for 1 to 2048 processors and task lengths from 1 to 32 seconds using a single dispatcher on a login node; Right: Efficiency graph for the Blue Gene/P for 256 to 160K processors and task lengths ranging from 1 to 256 seconds using N dispatchers with each dispatcher running on a separate I/O node

We see there are many workloads that operate efficiently at any scale (4 to 32 second long jobs at small scales, and 64 to 256 second long jobs at large scales), but there is certainly room for improvement. Specifically, it would be ideal if we can achieve high efficiencies (90%+) for any workload with work granularity in seconds, at any scale of existing systems (at least hundreds of thousands of processors). This involves improving performance by at least several orders of magnitude from the existing state of the art system. We expect to be able to build an execution fabric that will scale from millions to billions of processors, and handle throughputs of thousands to millions of jobs per second.

We envision building a new distributed data-aware execution fabric that scales to at least millions of processors and tens of petabytes of storage, and will support HPC, MTC, and HTC workloads concurrently and efficiently. Clients will be able to submit computational jobs into the execution fabric by submitting to any compute node (as opposed to submitting to single point of failure gateway nodes), the fabric will guarantee that jobs will execute at least once (jobs could be executed more than once under certain conditions such as unusual delays or transient failures), and that it will optimize the data movement in order to maximize processor utilization and minimize data transfer costs. The execution fabric will be elastic in which nodes will be able to join and leave dynamically, and data will be automatically replicated throughout the distributed system for both redundancy and performance.

To keep overheads small, and to have the system as portable as possible (e.g. to run on many of the top 10 of the Top500 supercomputers), we will use a general language such as C, general PThread libraries, proprietary communication protocols running over UDP/IP and/or TCP/IP, advanced data-transport protocols such as UDT that can achieve high performance over high latency links, and lightweight security libraries (e.g. ysSSL [70]) which have only KB of memory overhead per connection. We will also use structured distributed has tables (DHTs) in order to maintain efficient connectivity and membership information under extremely large scales of processor counts. The use of DHTs will allow us to efficiently find jobs and data files throughout the system in logarithmic time in relation to the size of the system. We will favor topologies that have many neighbors (e.g. 1000 closest neighbors), allowing us to traverse even large systems of millions of processors in only several hops (e.g. $\log(1000000)_{1000} = 2$). We will investigate various DHTs (e.g. Tapestry [71, 72], Chord [73, 74]) that have a certain set of desired properties. These properties include having a scalable light-weight implementation that is compatible with high-end computing systems (e.g. being implemented in C/C++, and having been tested on Linux). Other desirable properties include DHTs that take into account the natural network topology of many high-end computing systems (e.g. 3D torus/mesh, tree, etc) when assigning communication costs between compute nodes. This will help in choosing close neighbors, and to favor data communication with closer nodes; this also addresses in part heterogeneous environments, and could be used to run such an execution fabric across geographic locations.

We will build a system that has many queues distributed across all compute nodes/processors. There will be no ordering requirement on unrelated items placed in the queues, and therefore we can avoid expensive synchronization overheads among queues. However, we will support job dependencies, and the execution fabric will guarantee that dependent jobs are executed in the correct order to satisfy the data-flow dependency. Clients will be able to submit work to any queue, and each queue will have the choice of executing the work locally, or forwarding the work to another queue based on some function it is optimizing. This function can optimize in the simplest case load balancing. In a more complex case, it would optimize data movement, or perhaps both. We will also employ work stealing [66] techniques in order to achieve good load balancing under light load conditions. Jobs will be replicated throughout the system to ensure that node failures do not cause jobs to be lost; however, replicated jobs will only execute upon a failure of its original job, hence not wasting computational resources. Upon completion of a job, or failure of a job that doesn't involve a node failure, the replicated jobs would be removed from the system. The level of replication of jobs will determine the probability of losing jobs in the system, given some node failure model. We have shown in previous work [67] that the amount of replication can be tuned to achieve an expected level of mean-time-to-failure. In our prior work, the replication was at the node level, but it is directly applicable to job-level replication as it is relatively straight forward to replicate jobs and the costs are only incurred at the insertion in the execution fabric, as well as maintaining the state about the replicated jobs.

Finally, we plan to have high-level specification of jobs at the client (e.g. execute job_i from i=0..n), which would allow a client to easily inject large amounts of jobs into the execution fabric but delegating the unrolling of the iterating jobs to many remote distributed queues. We will also allow jobs to create new jobs effectively enabling dynamic DAGs to be easily be created and refined at runtime. Clients will also persist running jobs to persistent storage, in order to avoid having to keep in-memory state of all active tasks, which should allow clients to scale many orders of magnitude better than current in-memory approaches. The drawback to this approach will be the complexity and costs of moving job information back and forth between persistent storage and memory, and the performance penalty that might come with such movement. We believe new storage technologies such as SSDs could help us achieve the level of performance needed to sustain tens of thousands of job/sec throughputs from a single client.

Distributed Metadata Management and Storage Systems

Operations on parallel file systems (e.g. GPFS [45]) that involve metadata can be extremely inefficient at large scale. Early experiments on the BlueGene/P system (see Figure 5) shows the various costs for file create, directory create, and script invocation at scales from 256 processors to 16K processors. The times reported are the wall-clock time reported by the remote processor interacting with the parallel file system. Ideal performance would be to have a flat line, as close as possible to the black line indicating the overhead that Falkon has with no I/O. Notice that script invocation performs quite well, with little overhead. However, even a carefully tuned experiment which creates files or directories across many different directories (to avoid expensive locking mechanisms in GPFS) still has many seconds of overheads, and the overhead grows substantially up to 16K processors (a modest 10% of the overall system). If careful care is not taken to avoid lock contention by spreading files or directories over many directories, performance degrades rapidly, to the point where an operation (e.g. create directory) that took milliseconds on a single processor, takes over 1000 seconds at 16K processor scales. We want to achieve significantly lower overheads of accessing and modifying metadata at small scales, and have a relatively flat overhead line as we scale the system. We want to ensure that the metadata performance of the storage system is not the bottleneck of the system.

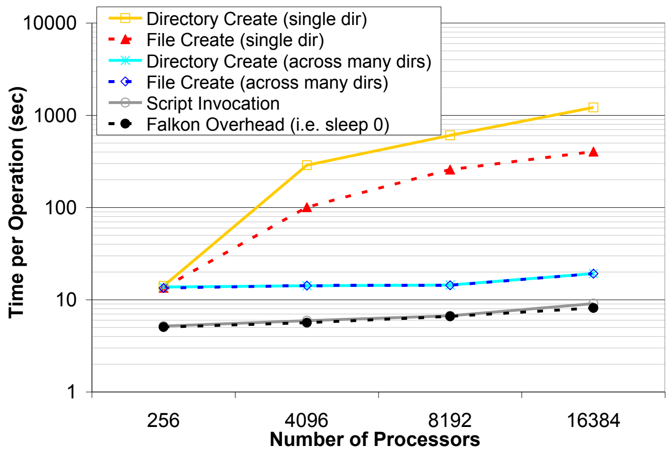
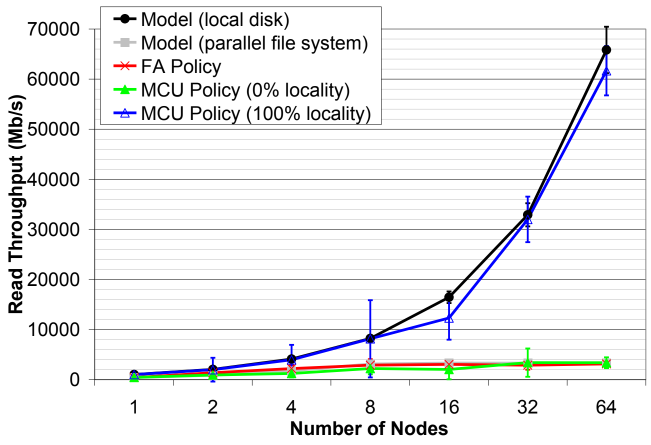


Figure 5: Time per operation (mkdir, touch, script execution) on GPFS on various number of processors (256-16384)

We plan to build a distributed storage system, which works at a file granularity, and has a global distributed hierarchical filename space. The architecture of the distributed storage system will be that every compute node will have all three roles, client, metadata server, and storage server. The metadata servers will be implemented on top of a distributed hash table, which will allow us to disperse the metadata information throughout the system, allow nodes to dynamically join and leave without compromising the metadata information, and allow us to locate files in the entire system in logarithmic time in relation to the size of the system (in the number of nodes).

Parallel file-systems are easy to use as they provide a single global name space which allows applications to easily manage and operate on large collections of data. However, they achieve this at a steep price, performance for the most demanding data-intensive applications, due to the fact that these file-systems hide data locality information from applications. Figure 6 shows a small cluster of 64 nodes performance between the GPFS parallel file system and the local disk performance of the 64 nodes. In both experiments (left and right figures), GPFS performance saturates at around 8 nodes (4Gb/s read), while local disk performance continues to grow almost linearly to 64 nodes (62Gb/s read). Now we have obtained performance of nearly 100Gb/s on large scale systems such as the BlueGene/P when using the GPFS parallel file system of that machine, but the theoretical amount of local I/O bandwidth on such a machine with 40K nodes is upwards of 40,000Gb/s (if each node can do a modest 1Gb/s to its local ramdisk). We believe we can harness a significant fraction of this bandwidth for workloads that have data locality [12], as we have shown in previous work [15, 16, 31].

Furthermore, future designs of large scale supercomputers will likely incorporate SSDs in with compute nodes, making the approach of distributing the data across many SSDs located on each compute node more attractive. The main concern for many HPC applications is the interference that might be caused by having some resources from the compute nodes and network interconnects be used for non-traditional I/O. In today's HPC systems, compute node resources are dedicated to the MPI-based application running, and the network interconnect is only used for MPI messages. However, in a distributed system such as the one presented in this work, compute nodes will also have the role of a metadata server and storage server, and some compute node resources would likely be consumed at all times on each node servicing requests on behalf of other compute nodes. This kind of variability in both the compute node resources and network interconnects might not be acceptable to some HPC applications, as it will cause their execution times to be increased significantly due to the imbalance created due to non-dedicated resources. Some of these concerns can be alleviated through dedicating certain resources (e.g. 1 core per node, 1 node per collection of 16 nodes) to handle the new functions. Furthermore, adding additional commodity and inexpensive network interconnects (Ethernet) could allow the traditional existing networks (e.g. torus, tree, barrier) to continue to be used exclusively for MPI-based workloads.



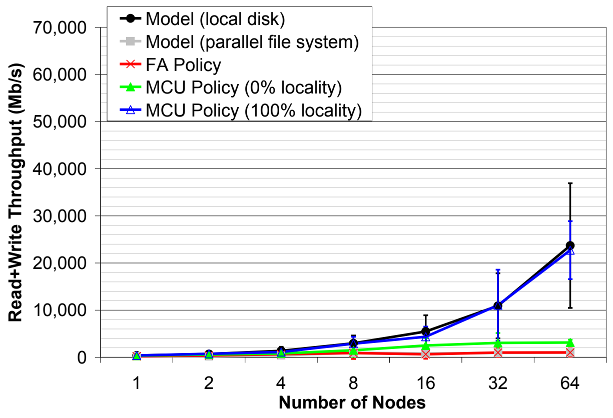


Figure 6: Left: Read throughput (Mb/s) for large files (100MB) for seven configurations for 1 – 64 nodes; Right: Read+Write throughput (Mb/s) for large files (100MB) for seven configurations and 1 – 64 nodes

We will implement a distributed storage system at the file level, which has a distributed metadata management, and a flat file namespace. The distributed file system will be accessible through library calls, and through a standard POSIX interface enabled by FUSE. As many current large-scale supercomputers do not have significantly amounts of local storage on the compute nodes, we will initially support a user-level approach that assumes to have an available parallel file system for persisting data, and the distributed storage system will be used for mainly scratch space to enable the highest performance for data-intensive applications (see Figure 7). Once large-scale systems have enough local storage (e.g. SSDs on each compute node), it is likely that such a system could stay online and be considered as a persistent storage system, just as parallel file systems are today.

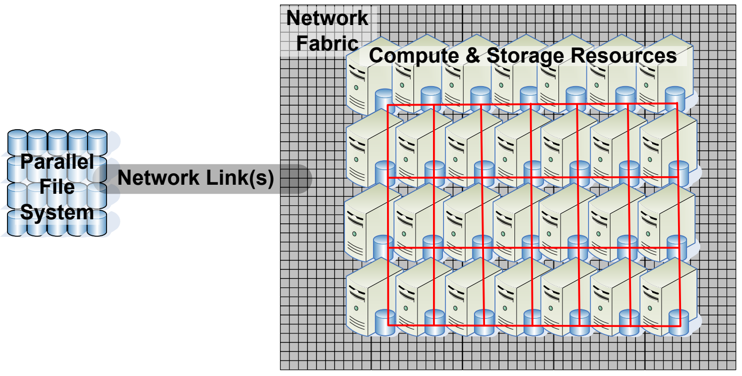


Figure 7: Overview of the proposed distributed storage system and distributed execution fabric

Distributed Data-Aware Scheduling

A challenge for data diffusion specifically has been scaling the centralized data-aware scheduler to extremely large systems with extremely large number of objects. Our micro-benchmarks have shown data diffusion to scale well to thousands of processors, but the data-aware scheduler becomes prohibitively expensive as we increase the number of processors to tens and hundreds of thousands. However, the non-data-aware simple load balancing scheduler parallelizes and performs well, as we have been able to scale it to 160K processors on the IBM Blue Gene/P running real scientific codes; due to the naïve implementation of the decentralized scheduler, it has utilization issues in heterogeneous systems and/or configurations.

Furthermore, data-aware scheduling is even more expensive/complex than the non-data-aware scheduling, and throughputs generally drop to about 1000 jobs/sec (see Figure 8) on a small cluster of only hundreds of processors; these throughputs can be maintained to thousands of processors, but beyond this the centralized data-aware scheduler costs begin to grow and the effective throughputs sink to single digit jobs/sec by tens of thousands of processors. We want the data-aware scheduler to be able to scale to millions of processors or more, and handle workloads with granularities on the order of seconds. This is many orders of magnitude higher performance than the existing data-aware scheduler in Falkon using data diffusion.

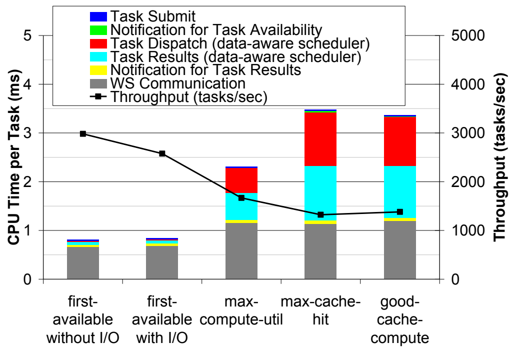


Figure 8: Data-aware scheduler performance and code profiling for the various scheduling policies

We will achieve this level of performance and scalability through the distribution of the data-aware schedulers to all compute nodes. Every compute node will keep track of a local cache of files, and will be able to query the execution fabric through the DHT about the location of needed files for tasks to be executed. The DHT will ensure that files will be able to be located in logarithmic time with the size of the system. While it was important at centralized data-aware scheduler to have high throughput in terms of jobs/sec, when the system has tens of thousands to millions of data-aware schedulers, it will be sufficient if each scheduler only performs single digit jobs/sec, as the aggregate system throughput will still be many orders of magnitude larger than the current state of the art.

Overlapping I/O with Computations

In previous work [38], we have successfully improved end-application performance by overlapping I/O to parallel file-systems with computations. Figure 9 shows the dramatic difference with 32 second long jobs, with different amounts of I/O per job, when overlapping the I/O with computations (the dotted lines) and when not overlapping them (the solid lines). Notice the significant difference in efficiency as the experiment scales to 96K processors with 1MB of I/O per job (5% efficiency vs. 90% efficiency).

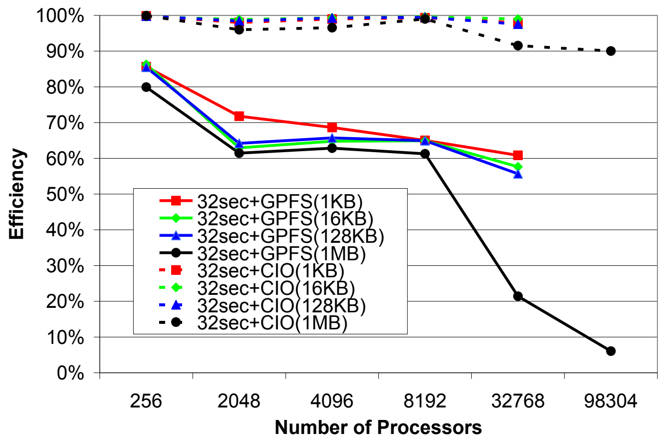


Figure 9: CIO vs GPFS efficiency for 32 second tasks, varying data size (1KB to 1MB) for 256 to 96K processors.

This suggests that I/O should be overlapped with computations whenever possible, and it should be done transparently without applications being modified. Essentially, we propose that the proposed distributed storage system allows applications to interact directly with the local file-systems which should allow write operations to occur at the fastest and most scalable speed. Once a data file is written locally, it would be the responsibility of the distributed storage system to propagate it asynchronously to other places in the distributed system where it might be needed, or perhaps persist the file to a parallel file system, while the respective compute node would be free to continue computing the next job(s). One was to achieve this transparency, which we have used in prior work [38], is to use symbolic links to make the application believe it is interacting with a persistent parallel file system, yet data is written to local scratch space, from which the distributed storage system will asynchronously move the data from local scratch to the persistent storage on a parallel file system. We can also implement this optimization to be completely transparent through the use of the user-level FUSE [68] interface that we can place on top of the proposed distributed storage system.

Generalization and Transparency

There are several road-blocks for wide-acceptance of a system like Falkon. One obstacle is the API Falkon exposes to applications that want to run on top of Falkon, essentially a web-service interface. Most of these applications already support other resource managers, such as PBS [51], Condor [36], SGE [52], Cobalt [47], and more abstract managers such as GRAM [69]. Adding support for Falkon is generally a straight forward exercise as the semantics of a job is the same in Falkon as it is in these other systems. One solution to avoid having each application adopt the new Falkon API, is to have adaptors in Falkon that allow Falkon to communicate with clients (applications) that were implemented to work with other existing resource managers. As most existing resource managers are open source, building support for those clients to interact with Falkon directly, using the other resource manager's APIs could be done at the expense of performance. It is likely that this approach would only be suitable for early testing and experimentation to allow non-Falkon enabled applications to take advantage of some of Falkon's innovative features and scalability. Every layer of abstraction/conversion adds more overhead, so the most streamlined approach will likely perform best, arguing in favor of applications adopting the new Falkon API to achieve the best performance possible.

Another obstacle Falkon has had with the data management component called data diffusion, has been the application hints about the files that are to be accessed. Some applications (e.g. those implemented through Swift) are naturally annotated with this information at run-time. However, other applications might be much more dynamic and would not know a-priori the files it needs to access (e.g. the file access patterns logic lies deep in the application, and is not configurable through application arguments); allowing these applications to still utilize the distributed storage system, even with limited data locality information, would be a big advantage. We plan to expose our distributed storage system through both I/O libraries and a POSIX filesystem through FUSE. We can even take this a step further, and consider the case where we the storage system could continuously compute the tradeoffs between transferring remote data to a local cache for processing, and saving the state of the application and re-scheduling it on the remote resource that had the needed data. This kind of process migration can be implemented through various virtualization technologies, or through application level check-pointing. We will seek to model data movement costs to accurately estimate under what conditions it will be justified to migrate a running job.

In summary, we'd like to achieve the same level of transparency that applications enjoy when running on top of existing production resource managers and existing parallel file-systems, but to achieve many orders of magnitude better performance and allow the efficient usage of tomorrow's largest distributed systems and supercomptuers.

2.2. Task 2: Interactive HPC

Support for MTC workloads combines traditional batch oriented HPC workloads with more MTC-like workloads that seeks fast response time on small tasks. With support for data-aware scheduling, coupled with a better handling of the memory hierarchy and I/O, we believe MTC could support interactive HPC (also known as ensemble MPI applications), enabling the execution of thousands of parallel computations each involving many thousands of cores and years of compute time in minutes of wall-clock time. Efficient support for interactive HPC can significantly alter the use of HPC in industry, as HPC applications that perform advanced simulations, parametric search and uncertainty quantification become part of an interactive design cycle, rather than an overnight validation cycle. One example is from the climate modeling domain, which has been studying climate trends and predicting global warming, is already implemented as an HPC MPI application. However, the current climate models could be run as ensemble runs (many separate concurrent MPI applications) to quantify climate model uncertainty in order to optimize the speedup and efficiency curves of the particular application (e.g. when scalability is poor). This is challenging in large scale systems such as supercomputers, a typical resource such models would execute on, as the local resource managers favor large jobs and have policy against running many jobs at the same time. Even if scheduling policies would not be the bottleneck, many production resource managers (e.g. Cobalt, SGE, PBS, Condor) have high scheduling and resource provisioning overheads which makes executing ensemble MPI applications only tractable if they are long running. This has implications in both the programming model, degree of parallelism, and reliability of application execution as the mean-time-to-failure (MTTF) generally decreases dramatically as applications and physical resources scale increase. All these applications pose significant challenges to traditional resource management found in HPC and HTC, from both job management and storage management perspective, and are in critical need of MTC support as the scale of these resources grows.

Ensemble MPI-based applications

We envision adding support for traditional HPC workloads to the existing Falkon middleware, such as support for MPI-based applications. We will support arbitrary resource allocation sizes, in contrast to other production systems that impose a limit on the smallest allocation size (e.g. for the IBM BlueGene/P at Argonne National Laboratory, the lower bound is 256 processors, and allocation sizes have to be multiples of 256). Furthermore, we will support sharing of data across different MPI applications with direct node-to-node communication, in contrast with having to go through some global file system, essentially leveraging all our work on data management from data diffusion. In the long-term, we will add support for HPC workloads on the proposed distributed execution fabric, which will allow us to scale these interactive HPC workloads (also known as ensemble MPI applications) to future exascale systems, as well as to leverage the distributed storage system infrastructure which will be critical for data-intensive HPC workloads.

Dynamic Steering of HPC workloads

Dynamic DAG based applications are inherently steerable at run-time, where the results of computations can alter the DAG's shape or branch that will be executed next. If DAG nodes can be composed of multi-processor MPI-based applications, in addition to the traditional single processor/node applications, then by definition we are able to support dynamic steering of HPC workloads.

3. Evaluation Strategy

We will perform an extensive performance evaluation of the execution fabric and the distributed storage system, using both micro-benchmarks and real applications. We will address various important metrics, such as scalability, throughput, speedup, efficiency, failure handling, endurance, robustness, I/O capabilities, and metadata performance. We will investigate a large class of applications, which fall under the following categories: bag-of-tasks, static and dynamic DAGs, all-pairs, MapReduce, and interactive HPC. We plan to use many tools to evaluate the proposed work, from simulations, to emulation, to real testing. We have access to five of the top eight supercomputers in the world (using the June 2009 Top500 rankings) from all of the big supercomputing vendors (Cray, SGI, IBM, and Sun). These five supercomputers represent the majority of supercomputer architectures and are ideal for research and development of novel data management techniques at the frontier of the largest scales available today. We also have access to the TeraGrid, the largest national cyberinfrastructure for open science research, as well as the Open Science Grid. Both of these grids offer unique opportunities to explore data management techniques that not only optimize data movement within tightly coupled systems, but also across geographic distribution that brings an entire new range of issues and challenges. We plan to do our performance evaluation up to the largest systems available today, up to 200K processors. Beyond this, we will use emulation to test system performance and scalability up to millions and tens of millions of emulated processors. We will further use simulations to evaluate our proposed work up to billions of processors and exascale of storage. We will also address the scalability and performance of our system through analytical models, and prove that our scheduling algorithms are efficient within a certain factor of ideal. We already have theoretical results for our centralized data-aware scheduler [15, 16], but we intend to extend that analysis to the distributed data-aware scheduler as well.

Task 1: Distributed Execution Fabric Performance Evaluation

Scalability, Throughput, Speedup and Efficiency, Failure Handling, Endurance and Robustness

Task 2: Distributed Storage System Performance Evaluation

Scalability, Characterizing I/O Capabilities, Failure Handling, Endurance and Robustness, Metadata Performance

Task 3: Applications Performance Evaluation

Bag-of-Tasks benchmarks, DAG-based benchmarks: Static and Dynamic, All-Pairs benchmarks, MapReduce benchmarks: TeraSort, PetaSort, WordCount, Interactive HPC benchmarks

4. Project Milestones

The timeline of proposed activities can be broken down into several major stages: explore, implement, evaluate, and disseminate. Note that some of these stages can be overlapped, and the end-to-end time needed to complete all proposed work is about 36 months.

- Explore: data-aware scheduling algorithms, solutions for intercepting I/O calls, file based and block level semantics, distributed file systems [12 months]
- Implement: distributed execution fabric and data-aware scheduler, I/O interception, POSIX file access to remote storage, MPI-support [24 months]
- Evaluate: micro-benchmarks of the data-aware execution fabric, interactive HPC workloads, real scientific codes before/after performance [12 months]
- Disseminate: document implementation and produce user guides, publish in top venues (e.g. SC, HPDC, NSDI, JGC, TPDS) [12 months]

5. Dissemination

I will disseminate the results of this work through various papers/proposals I plan to write this coming academic year:

- 11/06/09: Middleware for Many-Task Computing, Cluster Computing Journal
- 12/18/09: Topic TBD (perhaps preliminary ideas on a distributed file system based on a structured DHT), USENIX Workshop on Peer-to-Peer Systems (IPTPS) 2010
- 01/15/10: Interactive HPC, ACM HPDC 2010
- 01/15/10: Distributed Data Management, Journal of Parallel and Distributed Computing 2010, Special Issue on Data Intensive Computing
- 01/24/10: Cluster Computing on top of GPUs (high level ideas discussed with Nikos and Gokhan), USENIX Workshop on Hot Topics in Parallelism (HotPar) 2010
- 04/10: Topic TBD, IEEE/ACM SuperComputing 2010
- 04/10: Data Intensive Many-Task Computing, NSF Hecura
- 05/03/10: Efficient moving of large datasets, TeraGrid 2010

I also plan to attend various conferences and workshops, if scheduling permits and there are enough funds:

- 10/20/09: Chicago IL
  - Cloud Computing and its Applications (CCA) 2009
- 11/14/09 - 11/20/09: Portland OR
  - IEEE/ACM Supercomputing 2009
  - ACM Workshop on Many-Task Computing on Grids and Supercomputers (MTAGS) 2009, co-located with Supercomputing 2009
- 02/22/10 - 02/23/10: Washington DC
  - CRA Career Mentoring Workshop
- 04/28/10 - 04/30/10: San Jose CA
  - USENIX Symposium on Networked Systems Design and Implementation (NSDI) 2010
  - USENIX Workshop on Peer-to-Peer Systems (IPTPS) 2010
- 06/14/10 - 06/15/10: Berkeley CA
  - USENIX Workshop on Hot Topics in Parallelism (HotPar) 2010
- 06/21/10 - 06/25/10: Chicago IL
  - ACM High Performance Distributed Computing (HPDC) 2010
- 08/02/10 - 08/05/10: Pittsburgh PA
  - TeraGrid 2010

In the past, we have organized various workshops, bird-of-feather sessions, and journal special issues that were highly relevant to the proposed work. These activities include:

- ACM Workshop on Scientific Cloud Computing (ScienceCloud), 2010; http://dsl.cs.uchicago.edu/ScienceCloud2010/
- IEEE Transactions on Parallel and Distributed Systems, Special Issue on Many-Task Computing, 2010; http://dsl.cs.uchicago.edu/TPDS_MTC/
- ACM Workshop on Many-Task Computing on Grids and Supercomputers (MTAGS), 2009; http://dsl.cs.uchicago.edu/MTAGS09/
- Cloud Computing and Its Applications (CCA) 2009; http://www.cca09.org/
- IEEE Workshop on Many-Task Computing on Grids and Supercomputers (MTAGS), 2008; http://dsl.cs.uchicago.edu/MTAGS08/
- Megajobs: How to Run One Million Jobs, BOF at IEEE/ACM Supercomputing 2008; http://gridfarm007.ucs.indiana.edu/megajobBOF/index.php/Main_Page
- Cloud Computing and Its Applications (CCA) 2008; http://www.cca08.org/

We plan to continue organizing the CCA, MTAGS, and ScienceCloud workshops as long as there is interest from the community. If interest from the community grows significantly, we will consider expanding these workshops to stand-alone conferences.

6. Educational Aspects

Jeanette Wing had a famous quote in 2006 -- *"**Computational thinking** will be a fundamental skill used by everyone in the world by the middle of the 21st Century."* -- which we have adopted as a pivotal concept behind our teaching philosophy. Computational thinking is the combination of analytical thinking commonly found in mathematics with fundamental computer science skills (e.g. abstraction, codification, layering) while emphasizing interdisciplinary scientific computing. We believe that over the course of the next decades, computer science will spread to all disciplines and become an applied computing discipline. As computer science educators, we need to facilitate the right curriculum, to educate everyone from all disciplines about computational thinking. Interdisciplinary teaching needs to be made the de facto standard, with fundamentals of computer science being taught at all levels of education from primary school, undergraduate and graduate education, spanning many years to allow the computational thinking we take for granted in computer science to be absorbed and become second nature. Teaching of computing typically boils down to just information technology and how to use every day software such as word processors or spread sheets; these are merely tools with significantly less potential of impact than if computational thinking were a core part of all curricula. We will work with other faculty at the university level to help define and implement the computational thinking across all disciplines.

We plan to **enhance undergraduate education** by introducing research early into the undergraduate computer science curriculum and encouraging minorities and underrepresented to do research. Through research, young and diverse students will learn important skills early in their undergraduate studies: teamwork, written and oral skills, computational thinking, interdisciplinary skills, and experimental skills. Furthermore, participation in research groups can promote retention by increasing personal attachment to the research group, research objectives and research advisor, and captivate more students to pursue graduate degrees. We plan to establish an interdisciplinary undergraduate research program with computational thinking at its core, which will extend our teaching goals to use research as a primary tool for education, and promote computational thinking to the younger generation.

Our proposed work is **extremely important** in the future of many-core computing, a revolution that is currently underway, which will put thousands of processors/cores/threads in every desktop in the next decade. Current programming paradigms from workstations with several processor/cores do not map well to these highly parallel future systems. The techniques and middleware we are proposing that will work at extreme scales of millions of processors will surely be applicable to the modest scalability of future many-core systems. We believe data-flow driven programming languages that decompose its work in DAGs are quite easy to represent in programming languages, yet they offer a powerful abstraction that maps well to highly parallel architectures. Educating today's students to understand how to use these emerging programming paradigms is critical to having an educated workforce in the coming decade when many-core systems will be common-place.

We **conclude** with an ancient proverb by the Chinese philosopher Confucius that is still very much true today in the 21st century: "*If you think in terms of a year, plant a seed; if in terms of ten years, plant trees; if in terms of 100 years, teach the people.*" – Confucius, 551BC – 479BC. By teaching computational thinking to all disciplines and at all levels, far more people will be computationally literate. This will enable new kinds of science and a new economic era of science-based innovation that could dwarf the last decades of technology-based innovation. Just as our forefathers have etched in stone the many skills we all take for granted, our belief is that I can be at the forefront of these exciting times, and help bring computational thinking to all disciplines!

7. References

[1] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, I. Foster, "The Globus Striped GridFTP Framework and Server," IEEE/ACM SuperComputing (SC05), 2005

[2] I. Raicu, I. Foster, A. Szalay. "Harnessing Grid Resources to Enable the Dynamic Analysis of Large Astronomy Datasets", IEEE/ACM SuperComputing (SC06), 2006

[3] I. Raicu, I. Foster. "Harnessing Grid Resources to Enable the Dynamic Analysis of Large Astronomy Datasets", GSRP, Ames Research Center (ARC), NASA, 2006 -- Award funded 10/06 - 9/07

[4] I. Raicu, I. Foster. "Harnessing Grid Resources to Enable the Dynamic Analysis of Large Astronomy Datasets: Year 1 Status and Year 2 Proposal", GSRP, ARC, NASA, 2007 -- Award funded 10/07 - 9/08

[5] I. Raicu, I. Foster. "Harnessing Grid Resources to Enable the Dynamic Analysis of Large Astronomy Datasets: Year 2 Status and Year 3 Proposal", GSRP, ARC, NASA, 2008 -- Award funded 10/08 - 9/09

[6] I. Raicu, C. Dumitrescu, I. Foster. "Dynamic Resource Provisioning in Grid Environments", TeraGrid Conference (TG07), 2007

[7] I. Raicu, Y. Zhao, C. Dumitrescu, I. Foster, M. Wilde. "Falkon: a Fast and Light-weight tasK executiON framework", IEEE/ACM SuperComputing (SC07), 2007

[8] I. Raicu, Y. Zhao, C. Dumitrescu, I. Foster, M. Wilde. "Falkon: A Proposal for Project Globus Incubation", Globus Incubation Management Project, 2007 – Proposal accepted 11/10/07

[9] I. Raicu, Y. Zhao, I. Foster, M. Wilde, Z. Zhang, B. Clifford, M. Hategan, S. Kenny. "Managing and Executing Loosely Coupled Large Scale Applications on Clusters, Grids, and Supercomputers", Extended Abstract, GlobusWorld08, part of Open Source Grid and Cluster Conference 2008

[10] I. Raicu, Z. Zhang, M. Wilde, I. Foster, P. Beckman, K. Iskra, B. Clifford. "Towards Loosely-Coupled Programming on Petascale Systems", IEEE/ACM SuperComputing (SC08), 2008

[11] I. Raicu, I. Foster, Y. Zhao. "Many-Task Computing for Grids and Supercomputers", Invited Paper, IEEE Workshop on Many-Task Computing on Grids and Supercomputers (MTAGS08), 2008

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

[13] I. Raicu, Y. Zhao, I. Foster, A. Szalay. "A Data Diffusion Approach to Large Scale Scientific Exploration", Extended Abstract, Microsoft Research eScience Workshop (MSES07) 2007

[14] Q.T. Pham, A.S. Balkir, J. Tie, I. Foster, M. Wilde, I. Raicu. "Data Intensive Scalable Computing on TeraGrid: A Comparison of MapReduce and Swift", TeraGrid Conference (TG08) 2008

[15] I. Raicu, I. Foster, Y. Zhao, P. Little, C. Moretti, A. Chaudhary, D. Thain. "The Quest for Scalable Support of Data Intensive Workloads in Distributed Systems", to appear at ACM HPDC09

[16] I. Raicu, I. Foster, Y. Zhao, A. Szalay, P. Little, C.M. Moretti, A. Chaudhary, D. Thain. "Towards Data Intensive Many-Task Computing", book chapter in Data Intensive Distributed Computing: Challenges and Solutions for Large-Scale Information Management, IGI Global Publishers, 2009

[17] Y. Zhao, I. Raicu, I. Foster, M. Hategan, V. Nefedova, M. Wilde. "Realizing Fast, Scalable and Reliable Scientific Computations in Grid Environments", Grid Computing Research Progress, Nova Publisher 2008

[18] I. Foster, Y. Zhao, I. Raicu, S. Lu. "Cloud Computing and Grid Computing 360-Degree Compared", IEEE Grid Computing Environments (GCE08) 2008

[19] I. Raicu. "Many-Task Computing: Bridging the Gap between High Throughput Computing and High Performance Computing", Doctorate Dissertation, Computer Science Department, University of Chicago, March 2009

[20] I. Raicu. "Many-Task Computing: Bridging the Gap between High Throughput Computing and High Performance Computing", ISBN: 978-3-639-15614-0, VDM Verlag Dr. Muller Publisher, 2009

[21] M. Wilde, I. Raicu, A. Espinosa, Z. Zhang, B. Clifford, M. Hategan, K. Iskra, P. Beckman, I. Foster. "Extreme-scale scripting: Opportunities for large task-parallel applications on petascale computers", Scientific Discovery through Advanced Computing Conference (SciDAC09) 2009

[22] M. Wilde, I. Foster, K. Iskra, P. Beckman, Z. Zhang, A. Espinosa, M. Hategan, B. Clifford, I. Raicu. "Parallel Scripting for App lications at the Petascale and Beyond", IEEE Computer Nov. 2009 Special Issue on Extreme Scale Computing, 2009

[23] A. Gara, et al. "Overview of the Blue Gene/L system architecture", IBM Journal of Research and Development 49(2/3), 2005

[24] IBM BlueGene/P (BG/P), http://www.research.ibm.com/bluegene/, 2008

[25] J. Ousterhout, "Scripting: Higher Level Programming for the 21st Century", IEEE Computer, March 1998

[26] Y. Zhao, I. Raicu, I. Foster. "Scientific Workflow Systems for 21st Century e-Science, New Bottle or New Wine?", IEEE Workshop on Scientific Workflows 2008

[27] J. Dean, S. Ghemawat. "MapReduce: Simplified data processing on large clusters." In OSDI, 2004

[28] 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

[29] I. Raicu, Y. Zhao, C. Dumitrescu, I. Foster, M. Wilde. "Falkon: A Fast and Lightweight Task Execution Framework", IEEE/ACM SC, 2007

[30] E. Deelman et al. "Pegasus: A Framework for Mapping Complex Scientific Workflows onto Distributed Systems", Scientific Programming Journal 13(3), 219-237, 2005

[31] I. Raicu, Y. Zhao, I. Foster, A. Szalay. "Accelerating Large-Scale Data Exploration through Data Diffusion", ACM International Workshop on Data-Aware Distributed Computing 2008

[32] M. Isard, M. Budiu, Y. Yu, A. Birrell, D. Fetterly. "Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks", European Conference on Computer Systems (EuroSys), 2007

[33] 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

[34] M. Livny, J. Basney, R. Raman, T. Tannenbaum. "Mechanisms for High Throughput Computing", SPEEDUP Journal 1(1), 1997

[35] M. Flynn. "Some Computer Organizations and Their Effectiveness", IEEE Trans. Comput. C-21, pp. 948, 1972

[36] D. Thain, T. Tannenbaum, M. Livny, "Distributed Computing in Practice: The Condor Experience", Concurrency and Computation: Practice and Experience 17(2-4), pp. 323-356, 2005

[37] J. Appavoo, V. Uhlig, A. Waterland. "Project Kittyhawk: Building a Global-Scale Computer", ACM Sigops Operating System Review, 2008

[38] Z. Zhang, A. Espinosa, K. Iskra, I. Raicu, I. Foster, M. Wilde. "Design and Evaluation of a Collective I/O Model for Loosely-coupled Petascale Programming", IEEE Workshop on Many-Task Computing on Grids and Supercomputers (MTAGS08) 2008

[39] Top500, June 2008, http://www.top500.org/lists/2008/06, 2008

[40] T. Hey, A. Trefethen. "The data deluge: an e-sicence perspective", Gid Computing: Making the Global Infrastructure a Reality, Wiley, 2003

[41] SDSS: Sloan Digital Sky Survey, http://www.sdss.org/, 2008

[42] CERN's Large Hadron Collider, http://lhc.web.cern.ch/lhc, 2008

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

[44] Open Science Grid (OSG), http://www.opensciencegrid.org/, 2008

[45] F. Schmuck and R. Haskin, “GPFS: A Shared-Disk File System for Large Computing Clusters”, FAST 2002

[46] A. Bialecki, M. Cafarella, D. Cutting, O. O’Malley. “Hadoop: A Framework for Running Applications on Large Clusters Built of Commodity Hardware”, http://lucene.apache.org/hadoop/, 2005

[47] N. Desai. “Cobalt: An Open Source Platform for HPC System Software Research”, Edinburgh BG/L System Software Workshop, 2005

[48] J. Frey, T. Tannenbaum, I. Foster, M. Frey, S. Tuecke. “Condor-G: A Computation Management Agent for Multi-Institutional Grids”, Cluster Computing, 2002

[49] D.T. Moustakas et al. “Development and Validation of a Modular, Extensible Docking Program: DOCK 5”, J. Comput. Aided Mol. Des. 20, pp. 601-619, 2006

[50] D. Hanson. “Enhancing Technology Representations within the Stanford Energy Modeling Forum (EMF) Climate Economic Models”, Energy and Economic Policy Models: A Reexamination of Fundamentals, 2006

[51] 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, 4th Annual Linux Showcase & Conference, 2000

[52] W. Gentzsch, “Sun Grid Engine: Towards Creating a Compute Power Grid”, 1st International Symposium on Cluster Computing and the Grid, 2001

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

[54] G.B. Berriman, et al., “Montage: a Grid Enabled Engine for Delivering Custom Science-Grade Image Mosaics on Demand”, SPIE Conference on Astronomical Telescopes and Instrumentation. 2004

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

[56] US National Virtual Observatory (NVO), http://www.us-vo.org/index.cfm, 2008

[57] KEGG’s Ligand Database: http://www.genome.ad.jp/kegg/ligand.html, 2008

[58] PL protein library, http://protlib.uchicago.edu/, 2008

[59] NIST Chemistry WebBook database, http://webbook.nist.gov/chemistry/, 2008

[60] S.F. Altschul, W. Gish, W. Miller, E.W. Myers, D.J. Lipman. “Basic Local Alignment Search Tool”, J Mol Biol 215 (3): 403–410, 1990

[61] Computational Neuroscience Applications Research Infrastructure, http://www.ci.uchicago.edu/wiki/bin/view/CNARI/WebHome, 2008

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

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

[64] C. Moretti, J. Bulosan, D. Thain, and P. Flynn. “All-Pairs: An Abstraction for Data-Intensive Cloud Computing”, IPDPS 2008

[65] D. Bernholdt, S. Bharathi, D. Brown, K. Chanchio, M. Chen, A. Chervenak, L. Cinquini, B. Drach, I. Foster, P. Fox, J. Garcia, C. Kesselman, R. Markel, D. Middleton, V. Nefedova, L. Pouchard, A. Shoshani, A. Sim, G. Strand, and D. Williams, “The Earth System Grid: Supporting the Next Generation of Climate Modeling Research”, Proceedings of the IEEE, 93 (3), p 485-495, 2005

[66] James Dinan, D. Brian Larkins, P. Sadayappan, Sriram Krishnamoorthy, Jarek Nieplocha. "Scalable Work Stealing", IEEE/ACM Supercomputing 2009

[67] Nithin Nakka, Alok Choudhary. "Failure data-driven selective node-level duplication to improve MTTF in High Performance Computing Systems", High Performance Computing Symposium 2009

[68] M. Szeredi. File System in User Space. http://sourceforge.net/apps/mediawiki/fuse/index.php?title=Main_Page, 2009

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

[70] yaSSL, http://yassl.com/, 2009

[71] Ben Y. Zhao, Ling Huang, Jeremy Stribling, Sean C. Rhea, Anthony D. Joseph, and John D. Kubiatowicz. "Tapestry: A Resilient Global-Scale Overlay for Service Deployment", IEEE Journal on Selected Areas in Communication, VOL. 22, NO. 1, January 2004

[72] Tapestry, http://current.cs.ucsb.edu/projects/chimera/, 2009

[73] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan, Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications, ACM SIGCOMM 2001, San Deigo, CA, August 2001, pp. 149-160

[74] Chord, http://pdos.csail.mit.edu/chord/, 2009

[75] ASC / Alliances Center for Astrophysical Thermonuclear Flashes, http://www.flash.uchicago.edu/website/home/, 2008

[76] Catalin Dumitrescu, Ioan Raicu, Ian Foster. "Experiences in Running Workloads over Grid3", The 4th International Conference on Grid and Cooperative Computing (GCC 2005)

[77] Catalin Dumitrescu, Ioan Raicu, Ian Foster. "The Design, Usage, and Performance of GRUBER: A Grid uSLA-based Brokering Infrastructure", International Journal of Grid Computing, 2007

[78] Ioan Raicu, Catalin Dumitrescu, Matei Ripeanu, Ian Foster. "The Design, Performance, and Use of DiPerF: An automated DIstributed PERformance testing Framework", International Journal of Grid Computing, Special Issue on Global and Peer-to-Peer Computing, 2006