ExM Progress Report

Daniel S. Katz

1 Project Information

DOE award number	ER26012
Name of the recipient	University of Chicago
Project title	ExM: System support for extreme-scale, many-task applications
Name of the project principal investigator	Daniel S. Katz
Date of report	May 31, 2011
Period covered by this report	September 1, 2010 – April 30, 2011

2 Introduction

The ever-increasing power of supercomputer systems is both driving and enabling the emergence of new problem-solving methods that require the efficient execution of many concurrent and interacting tasks. Methodologies such as rational design (e.g., in materials science), uncertainty quantification (e.g., in engineering), parameter estimation (e.g., for chemical and nuclear potential functions, and in economic energy systems modeling), massive dynamic graph pruning (e.g., in phylogenetic searches), Monte-Carlo- based iterative fixing (e.g., in protein structure prediction), and inverse modeling (e.g., in reservoir simulation) all have these requirements. These many-task applications frequently have aggregate computing needs that demand the

fastest computers. For example, proposed next-generation climate model ensemble studies will involve 1,000 or more runs, each requiring 10,000 cores for a week, to characterize model sensitivity to initial condition and parameter uncertainty.

The goal of the ExM project is to achieve the technical advances required to execute such many-task applications efficiently, reliably, and easily on petascale and exascale computers. In this way, we will open up extreme-scale computing to new problem solving methods and application classes.

In this document, we report on combined technical progress of the collaborative ExM project, and the institutional financial status of the portion of the project at University of Chicago, over the first 8 months (through April 30, 2011)

3 Year 1 Accomplishments Summary and Schedule Status

The objectives below are those listed as for Year 1 in Section 8 of the original ExM proposal.

3.1 Aim 1: ExM data store development and component-level testing

3.1.1 Objective 1.1

Detailed analysis of bottlenecks and potential performance optimization opportunities for target applications. Detailed analysis of capabilities of target deployment platforms.

Fraction Complete: 70%

Notes: See §5.1.1

3.1.2 Objective 1.2

Design of virtual data store prototype including: ability to support specialized APIs (MPI-IO, HDF5) in addition to POSIX; ability to support cross

layer optimizations through custom metadata.

Fraction Complete: 50%

Notes: See §5.1. Currently our efforts are focused on the POSIX interface.

3.1.3 Objective 1.3

Implementation of data store prototype that: can be instantiated on-the-fly at application launch time, integrates memory and disk based resources provided by the application, supports POSIX API, supports a limited number custom tags for performance optimizations (data prefetching, custom replication levels, data lifetime directives, file intensity access directives).

Fraction Complete: 50%

Notes: See §5.1

3.1.4 Objective 1.4

Target scaling level: 1,000 nodes and clients demonstrated in application driven scenarios.

Fraction Complete: 70%

Notes: See §5.1.3

3.1.5 Objective 1.5

Unit based testing and overhead evaluation using microbenchmarks.

Fraction Complete: 70%

Notes: We have increased the Unit Test coverage in MosaStore. See also: §5.1.2

3.2 Aim 2: ExM Task Graph development and testing

3.2.1 Objective 2.1

Design and implement initial core fast unified task evaluator and queue manager, and in-memory function call / data structure model, integrating Swift,

ADLB, and Falkon.

Fraction Complete: 50%

Notes: Developed infrastructure to deploy distributed many-task scheduler on IBM Blue Gene/P (BG/P) over ZeptoOS. Developed and reported on rapid task launch infrastructure for MPI tasks (JETS). Performed performance analysis using ADLB to launch Falkon/Swift-like tasks on the BG/P. Developed sequential prototype of new task dependency graph manager based on location-independent execution model.

3.2.2 Objective 2.2

Features will include static graph partitioning with program/API hints, multilevel graph partition executor, fast RAM-based task queue executor, initial integration with ExM data manager, and scheduler integration and MPI application handling.

Fraction Complete: 25%

Notes: Initial development of intermediate user code to be used by distributed task manager. Developed MPICH management hooks for MPI application handling.

3.2.3 Objective 2.3

Develop synthetic benchmark suite; test on ALCF BG/P, OLCF XT5, TeraGrid Constellation & XT5.

Fraction Complete: 75%

Notes: Task rate measurement scripts and plot generation tools are in place. Performance tests have been performed on the BG/P and Cray systems. Ongoing work will apply these scripts to developing ExM methods and systems.

3.2.4 Objective 2.4

Scaling targets: 1M node graph on 100K cores, 60 second tasks with 85% efficiency on Intrepid BG/P. Will apply for INCITE allocations and Blue Waters PRAC (or Blue Waters Director's Discretionary allocation) and maintain current TeraGrid/Ranger/Kraken and ALCF discretionary allocations.

Notes: Scaling target not yet accomplished. Allocations on appropriate resources have been maintained and used by the the project. ExM is in active communication with the Blue Waters leadership.

3.3 Aims 3 and 4: System integration and Application integration and evaluation

We will plan to release code snapshots twice per year, with latest code continually available to user community via Subversion.

3.3.1 Objective 3.1

Integrate & harden software from Data Store and Task Graph teams.

Fraction Complete: 10%

Notes: We've started to do this, but the software development has had a slower than expected start.

3.3.2 Objective 3.2

Demonstrate one full stack application workflow (e.g., Montage) integrating Application/Swift/Falkon/MosaStore operating over thousands of compute nodes on one large-scale system.

Fraction Complete: 50%

Notes: We plan to demonstrate ModFTDock, and have been working with additional applications, as discussed in §5.3

3.3.3 Objective 3.3

At least one conference paper will be submitted.

Fraction Complete: 100%

Notes: See §10.1 - we have three accepted conference papers based at least in part on this work, and one more that is currently under review.

4 Summary comparison of accomplishments and goals/objectives

We have made good progress towards the objectives listed in Section 3, given that we are working at 60% funding, and that this report covers just 75% of the first year.

Regarding the funding, these objectives were written for a project with a budget of $2.86 over 3 years, while the funding we have received is $1.8m over 3 years.

5 Technical Details of Accomplishments

5.1 The ExM Data Store

We have explored two data store designs: the MosaStore (`http://mosastore.net/`) and the AME (Any-scale Many-task computing Engine) intermediate data store. While with both systems we aim to offer a high-performance, reliable data-store, the crucial difference between these two systems is that MosaStore aims to explore the bounds of obtainable performance when the storage system design is constrained to offer a POSIX API, whereas the AME intermediate data store accepts a complete departure from the constraints of the POSIX API at the cost of application redesign to a new programming interface.

The rest of this section briefly presents these two systems and our main progress to date.

5.1.1 The AME Data Store Introduction

In the AME intermediate data store, metadata is fully distributed to a group of servers by a consistent hash function. With a distributed memory coherence protocol, job dependencies can be resolved in a distributed manner. Rather than storing the data itself in the DHT, we only store the metadata and location mapping in the DHT servers. The data can be transferred once the worker determines which remote site actually holds the data.

In our AME work, the bottlenecks are categorized as Resource Provisioning, Job Dispatching, Load Balancing, Data Management, and System Resilience. Among those bottlenecks, Job Dispatching, Load Balancing, Data Management and System Resilience are being addressed in ExM research, as Resource Provisioning usually involves in policy issues of the institutions who own the hardware. To solve those bottlenecks, we conducted a cross-platform study of existing parallel and distributed computing tools.

Job Dispatching: MPI leaves this function to programmers. Commonly, MPI compiles the list of tasks, which is loaded by all compute nodes, and each compute node that does its part of the work as identified by the worker's rank. This scheme is the most scalable of the ones we covered, but it requires the compute nodes to load redundant information. Every compute node needs to load the compiled binary, and finds out its own task. Pegasus/Condor uses a centralized job dispatcher, the submit host, which keeps a shadow of every single job. It tracks lifetime state change of the jobs. Thus its scalability is limited to the capacity of the submit host. Also, it consumes a lot more memory than the MPI case. Falkon explores a 3-tier architecture: a first tier submit host, a group of second tier dispatchers, and a group of third tier workers. Nevertheless, Falkon's job view is the same as Condor, and limits the scalability of short jobs (O(1) s). AME's dispatcher takes advantage of Falkon's 3-tier design, and changes the job view from a single job to a text file at the first layer. AME's dispatcher does not monitor job status, which results in higher scalability than Falkon; it is a tradeoff between scalability and the detailed job status monitoring.

Load Balancing: The MPI standard does not provide such functionality; ADLB (and other higher level schemes) aims to solve this bottleneck in the MPI scenario. In the parallel programming languages Cilk, and Parlog, researchers have put effort in the work-stealing algorithm that tries to remedy the starving situation that may occur. But none of the existing work-stealing algorithms has ever been proved to scale up to 100,000 cores. AME wants to address this problem via a local stealing scheme, where each compute node cannot access every other compute node in the allocation. This is because the Torus network is a hop-by-hop transmission network, so by limiting the stealing scope, we can avoid high-latency round-trips. Also, this can balance the stealing workload across all the compute nodes in the allocation.

Data Management: ROMIO is designed as the I/O support for MPI, which can also be viewed as the Data Management Module. MPI also leaves

job length (sec)	256 cores	512 cores	1024 cores	2048 cores	4096 cores	8192 cores	16384 cores
1	66.39%	65.79%	65.84%	66.12%	65.60%	65.23%	64.88%
4	89.39%	89.37%	89.37%	88.86%	89.21%	89.14%	87.94%
16	97.03%	97.12%	97.11%	97.08%	97.06%	97.01%	96.82%
64	99.27%	99.26%	99.26%	99.25%	99.25%	99.12%	98.87%

Table 1: Linear Scalable Efficiency of 1,4,16,64,256 seconds job length without data

this feature to programmers. In most MapReduce scenarios, the data to be processed are assumed to reside on the compute nodes. HDFS (Hadoop Distributed File System) places 3 replicas of each data chunk over the compute nodes. Other work tries to isolate the data storage and processing. HDFS's scalability is mainly limited by its single management node architecture. Results show it scales up to the level of 1,000 compute nodes, but not more. There are two branches to build the data store. One is MosaStore, which is a chunk based, storage and processing isolated data store. The other one is the AME intermediate data store, which is file based. It employs a group of dedicated metadata servers that are running on top of a consistent DHT servers for scalability purpose. As it is file based, the file size limit will be bounded by the ramdisk size of compute nodes. MosaStore does not have this problem, as the file size limit is subject to the aggregated ramdisk size.

System Resilience: MPI does not provide any resilience features: when an MPI program fails, the user needs to restart the run. Condor uses a checkpointing scheme for resilience. MapReduce duplicates jobs as it takes node failure as a normal situation rather than an exception in the commodity clusters. We are still working on how AME should address this issue.

5.1.2 AME Unit Test

Table 1 shows AME job dispatching performance without data. In the test setup, each core runs 16 jobs with an identical run time. The ideal time to solution is 16 seconds, 64 seconds, 256 seconds, 1024 seconds, respectively,

job length(sec)	256 cores	512 cores	1024 cores	2048 cores	4096 cores	8192 cores
1	24.14%	23.66%	24.99%	24.24%	22.07%	21.23%
4	70.86%	70.64%	70.53%	70.84%	67.38%	63.57%
16	93.86%	91.66%	91.38%	91.87%	90.20%	89.14%
64	97.61%	98.08%	98.00%	97.66%	98.22%	97.10%

Table 2: Linear Scalable Efficiency of 1,4,16,64,256 seconds job length with data

file size(byte)	256 cores	512 cores	1024 cores	2048 cores	4096 cores	8192 cores
1K	90.77%	90.00%	89.88%	88.45%	87.00%	86.45%
1M	91.04%	87.76%	88.99%	88.32%	86.66%	86.03%
10M	87.92%	88.41%	86.64%	86.36%	86.06%	83.87%

Table 3: Linear Scalable Efficiency of 16 seconds job length with data size of 1 KB, 1 MB, and 10 MB

for job lengths of 1 second, 4 seconds, 16 seconds, and 64 seconds.

Table 2 shows AME performance with intermediate data store. In the test setup, each core runs 16 jobs. The 16 jobs are divided into 8 pairs, where each pair of jobs has a file dependency. The file size in this test is 10 bytes. The ideal time to solution is 16 seconds, 64 seconds, 256 seconds, 1024 seconds, respectively, for job lengths of 1 second, 4 seconds, 16 seconds, and 64 seconds.

Tables 1 and 2 show that the AME system features linear scalability for both job dispatching and intermediate data management. Table 3 shows that using 10 MB files has a ∼2-3% overhead, and at the 8,192-core scale have a ∼3% overhead compared to the 256-core scale.

	# of jobs	1 core (s)	512 cores (s)	speedup
mProject	1319	21220.32	56.53	375.38
mDiffFit	3883	35960.12	95.32	377.27
mBackground	1297	9815.92	64.44	152.33

Table 4: Performance Comparison of AME and Single-node execution

5.1.3 AME Scaling

We have scaled AME's intermediate data store up to 8,192 cores on ANL's IBM Intrepid with linear scalability. From the 256-core scale to 8,192-core scale test, there is 1% overhead for the 64-second job length with 8 rounds of data shuffling. Then, from the data size test, transferring 10-MB files places a overhead of 3.08% comparing to the 10-Byte file size at 8,192-core scale. We ran a test of Montage that produces a 6 x 6 mosaic centered at galaxy M101. It has 1,319 input files, each of which is 2 MB. Stage 1 outputs 1,319 4-MB files. We ran the 2nd and 5th stage with the AME built-in reduction function. Stages 3 and 6 runs on the login node, as they analyze summarized files, and generate new jobs. Stages 1, 4, 7 each run in a parallel manner; they process the input/output data with the data management scheme we described in previous sections. Each job in Stage 7 writes a file of 4-MB size. We compare the performance of the 512-core approach with a single node execution to show speedup, as in Table 4. The time is measured in seconds.

The 1-core data is estimated from the performance of the login node, which is 4 times faster than a compute node. The mBackground stage has a lower speedup because it moves the output from compute nodes to GPFS. If we can run mAdd in an MTC style, then we could reduce this consumption by transferring data among compute nodes, and only port the mAdd output to GPFS. The mImgtbl and mBgModel stages are done with the AME built-in reduction function. The processing times are short, 9.6 and 14 seconds respectively. In this test, we reduce the data movement from compute nodes to GPFS by 45.6 GB, as shown in Table 5.

	GPFS (MB)	AME (MB)	Saving(%)
mProject-input	2800	2800	0%
mProject-output	5500	0.36	100%
mDiffFit-input	31000	0	100%
mDiffFit-output	3900	0.81	100%
mBackground-input	5200	0	100%
mBackground-output	5200	5200	0%
total	53600	8001.17	85.72%

Table 5: Comparison of Data Transfer Amount between GPFS and AME Approaches

5.1.4 The MosaStore Data Store

MosaStore (`http://mosastore.net/`) is an experimental data-storage system. MosaStore is designed to harness unused storage space from network-connected machines to build a high-performance, yet low-cost datastore that can be easily configured for optimal performance in different environments and for different application data access patterns while still maintaining a POSIX API. MosaStore aggregates distributed storage resources: storage space (based on spinning disks, SSDs, or memory) as well as the I/O throughput and the reliability of the participating machines.

We have two strategic goals with MosaStore: First, MosaStore is meant to support full fledged applications that are deployed in contexts where aggregating resources into a specialized storage system is advantageous. For example, MosaStore can be used to support many-task applications, to provide a specialized, high-performance scratch space, to support checkpointing, or as a glide-in datastore. Second, we will use MosaStore as a platform to explore and evaluate innovations in the storage-system architecture, design, and implementation. For example, we plan to extend MosaStore to explore the feasibility of cross-layer communication through custom file metadata, explore solutions to automate storage-system configuration, to explore support for data deduplication, to explore the feasibility of a versatile storage system. These are only a few of the advanced research projects that will exploit (and hopefully contribute) to the MosaStore codebase and, during the past year, we have made considerable progress in this direction (accepted

paper 3 in §10.1).

Currently we have a full prototype implementation of the system. So far we have tested this prototype with two scientific-workflow applications (modFTDock and Montage) on pure-Linux clusters and at a limited scale. Currently we are focusing our effort on supporting larger-scale execution on a wider class of platforms (e.g., Argonne's IBM BlueGene/P, Cray), incorporating key performance optimizations, and extending the number of applications supported.

It is worth highlighting a number of key MosaStore features or utilities we have already successfully incorporated: the ability to expose file location (through file extended attributes); a profiler to better understand the structure of all FUSE calls MosaStore serves; preliminary experience with evaluating the energy footprint of the MosaStore datastore; and experience with automating the configuration of the datastore to determine performance-optimal configurations.

An ongoing technical document describing MosaStore and providing details about the design, current implementation status and short- and mid-term plans is available here: `http://www.ece.ubc.ca/~matei/ExM.Internal/`

5.2 Task Graph

To support the development of the ExM distributed task manager, we have carried out the following investigations. The system is diagrammed in Figure 1 for reference.

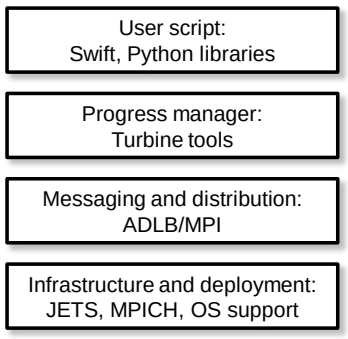

Figure 1: ExM components for workflow generation and deployment.

5.2.1 System infrastructure

The procedures involved in rapidly launching multiple concurrent MPI tasks were not previously supported on HPC systems such as the BG/P. To address this, we developed a lightweight infrastructure to deploy MPI programs over a network of worker agents. This mechanism, called JETS, enables very fast execution of small tasks on a variety of resources. The mechanism is consists of in-

tegration with ExM-led changes to the MPICH implementation and tools provided by the ZeptoOS project. In cooperation with application users of the popular NAMD molecular dynamics simulator, we demonstrated that the ExM application model is useful for scientific workloads (see Figure 2). However, JETS relies on a centralized component; additional distribution of the task management infrastructure is necessary.

5.2.2 Task distribution

ExM proposes to distribute user tasks through the The Asynchronous Dynamic Load-Balancing Library (ADLB) library, and MPI-based distributed master-worker scheme. We investigated task distribution rates through an ADLB-based program called `batcher`, which launches user commands across the compute infrastructure. We used JETS to deploy `batcher` and measured ADLB performance in multiple OS and communication modes on the BG/P, including a) native MPI on the IBM-provided kernel, b) native MPI on the ZeptoOS kernel, and c) MPI over TCP on the ZeptoOS kernel. Only method c) currently allows the use of `fork()`, which is required for our purposes. We found that a) and b) both perform extremely well and c) results in significant latencies, however, the imposed delays will not prevent us from reaching the scaling target.

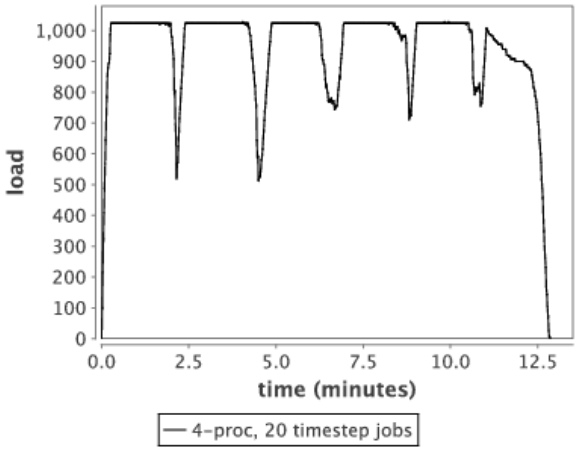

Figure 2: Results from many parallel task NAMD run in JETS on the BG/P. *Results show high utilization of the 1,024 nodes divided into 256 concurrent 4-node tasks. Dips in the load level are caused by task stop/start in the replica exchange algorithm.*

5.2.3 Language support and dependency management

The ExM team has begun the design of an intermediate code layer to support the distribution of the dependency management required for our target user

languages: Swift, and a Python-based library. The language implementations will generate the intermediate code, called the ExM Prototype Pseudo Instruction Code (EPPIC). Preliminary EPPIC instructions may be evaluated in the ExM Turbine library. Turbine is a small library to evaluate the dependencies generated the user script and drive the execution of the user application. The Turbine calls will shortly be integrated in the `batcher` program to enable data dependency-driven execution in and ADLB-like setting, which will thus enable the development of complete applications.

5.3 Integration and Applications

We have made a good start at working with various applications, specifically:

- ModFTDock – Protein Docking – Docking can identify molecules having high affinity towards a particular protein. This property can be used for selection processes in drug design. ModFTDock capabilities include docking RNA to protein molecules. This is done by providing a database of protein molecule's 3D spatial properties (including separation parameters) represented in the form of a multicolumnar layout through a '.pdb' file format.

- Montage – Astronomical Image Mosaicking – complex multi-stage workflow, medium data movement requirements, large amounts of intermediate data

- SCEC Cybershake – Seismic Hazard Assessment – very complex data movement, O(1m) tasks per run

- CIM-EARTH – climate and energy policy – a collaborative, multi-institutional project to design a large-scale integrated modeling framework as a tool for decision makers in climate and energy policy. CIM-EARTH is intended to enhance economic detail and computational capabilities in climate change policy models, and to nucleate and support a broad interdisciplinary and international community of researchers and policymakers. A complete run involves O(100k) tasks.

- Hybrid Multiscale Subsurface Simulations – simulations of hybrid coupling of pore- and continuum-scale flow and reactive transport models.

This application generates many MPI tasks, and is a good laboratory for hybrid MPI/many-task computing models.

- Parallel high-resolution climate data analysis. With an order of magnitude increase in the volume of output data produced by the Community Earth System Model (CESM), post-processing model output data has become a bottleneck in the analysis process. We are working, in collaboration with the ASCR project "ParVis" on a parallel many-task version of the CESM atmosphere model data analysis workflow using Swift.

- Projecting impacts on hydrology and energy development by improving regionally coupled hydrology and climate models. This application uses the SWAT hydrology simulation program in a multi-round genetic algorithm pattern, and has many challenging data-intensive characteristics. We have started working on expressing this application as a many-task Swift script and will continue working on it with an ExM summer student June-Aug 2011.

5.3.1 Application Analysis

We have made significant inroads towards analyzing the performance bottlenecks and optimization opportunities for the target application. We have gained insights into the low level operations and data exchange patterns through system-level profilers such as the *Darshan* framework developed at Argonne's MCS department. As an illustration of the above mentioned analysis, Figure 3 contains a series of plots showing the various application I/O patterns and associated costs for the ModFTDock application. Note that the MPI related operations are not the application specific in that they have been artificially injected in the application in order to trigger the Darshan profiler.

The analysis plots in Figure 3 give a view of low-level activities going on during a unit run of the modFTDock application. More specifically, we gain the following insights for each run of the application:

1. It takes 15 seconds to complete, of which about 1.9 seconds (13.2%) are spent in the reading of metadata operation (shown by the blue part

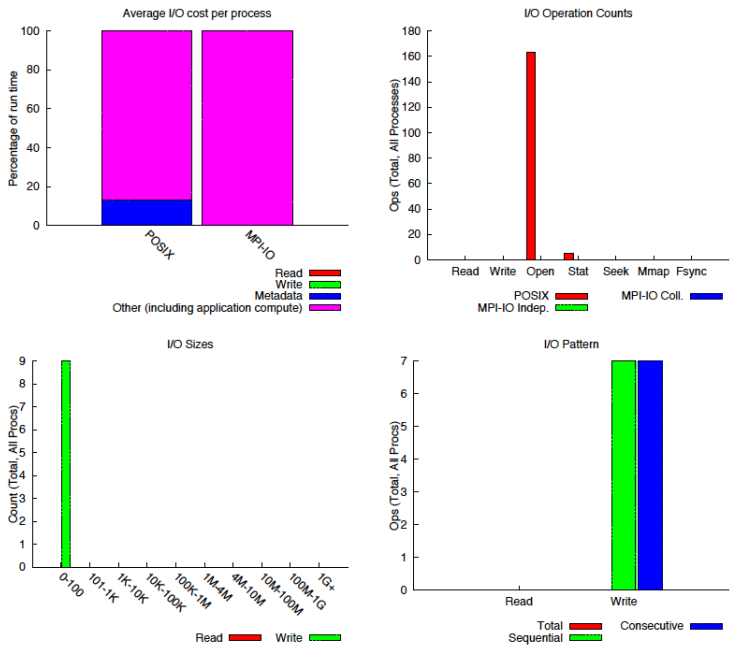

Figure 3: Darshan Plots for the ModFTDock Application

of top-left plot).

2. It involves about 180 I/O operations dominated by the POSIX `open` calls followed by the `stat` calls (shown by the top right plot).

3. It has 9 write operations, each writing data less than 100 Bytes (shown by the bottom left plot).

4. It has 7 sequential and consecutive write operations each (shown by the bottom right plot).

This basic information is useful to us as we plan how to create and run ensembles of the application, as it tells how the total I/O load would occur if all I/O simply used the shared file system, and it points us to where collective I/O and intermediate data storage will be needed.

6 Cost Status

As of 4/30/11, we have spent $33,900. If we continued at the current rate, at the end of year 1, we will have spent $103,512 and will have unspent funds of $119,497. We are underspent for two reasons:

1. We started spending late, due to staff transitioning from other projects.

2. Due to the reduced award, we decided to change the spending profile, and push some of the integration and applications work to start late in the first year rather at the start of the project, so we would be able to have a larger fraction of a staff member for the remainder of the project. (e.g., had we originally planned 0.75 FTE of a staff person on these activities over the full 3 years, with the reduced budget, rather than having 0.5 FTE staff over 3 years, we would have 0.65 FTE staff over 2.25 years)

We plan to increase our spending by increasing adding a large fraction of a staff member to work on Aims 3 and 4 starting June 1, 2011, for the remainder of the project.

7 Changes

There have been no changes in approach or aims. The only significant changes to the objectives and scope are those due to the reduced budget.

8 Problems

There have been no significant problems or delays, and none are currently anticipated.

9 Key personnel & consortium/teaming arrangement

There have been no absences or changes of key personnel or changes in the consortium/teaming arrangement.

10 Products & technology transfer activities

10.1 Publications

Accepted/Published:

1. Swift: A language for distributed parallel scripting. Michael Wilde, Mihael Hategan, Justin M. Wozniak, Ben Clifford, Daniel S. Katz, Ian Foster. *Parallel Computing Journal*, in press.
   PDF: `http://www.mcs.anl.gov/uploads/cels/papers/P1818.pdf`

2. Towards Automating the Configuration of a Distributed Storage System. Lauro B. Costa, Matei Ripeanu, *11th ACM/IEEE International Conference on Grid Computing (Grid 2010)*, Brussels, Belgium. October 2010.
   PDF: `http://goo.gl/BdV0f`

3. Assessing Data Deduplication Trade-offs from an Energy Perspective. Lauro Beltro Costa, Samer Al-Kiswany, Raquel Vigolvino Lopes, Matei Ripeanu, *Workshop on Energy Consumption and Reliability of Storage Systems (ERSS)*, Orlando, Florida, July 2011.
   PDF: `http://goo.gl/VYUJ7`

4. JETS: Language and System Support for Many Parallel Task Computing. Justin M. Wozniak and Michael Wilde, *Workshop on Parallel Programming Models and Systems Software for High-End Computing (P2S2) at International Conference on Parallel Processing (ICPP)*, Taipei, Taiwan, September 2011.
   PDF: `http://www.mcs.anl.gov/uploads/cels/papers/P1885.pdf`

Submitted:

1. AME: An Any-scale Many-task Computing Engine, Zhao Zhang, Ian Foster, Daniel Katz, Michael Wilde, Ioan Raicu, submitted to *SC11*, Seattle, Washington, November 2011.

Student work towards degrees:

1. Lauro Costa, Ph.D. proposal, for Ph.D. qualifying exam in June 2011: Automating the Configuration of Distributed Storage Systems

2. Zhao Zhang, Master's paper, for presentation in June 2011: Bridging the Gaps between MTC and Supercomputers.

10.2 Web site

Public web site: `https://sites.google.com/site/exmcomputing/` (Currently under construction)

Internal web site: `https://sites.google.com/site/exmproject/` (Credentials can be provided for DOE program managers on request)

10.3 Collaborations

We have collaborated with the following projects/personnel:

The current and anticipated results of the ExM project play a leading role in a proposed SciDAC Institute for intelligent dynamic ensemble applications (IDEA) that involves: Tim Germann (LANL, PI of the Extreme Materials Exascale Co-design Center), Jeff Hammond (Argonne, co-PI of the Chemistry Exascale Co-design Center), Rob Jacob (Argonne, PI of the DOE-BER climate data analysis and visualization project ParVis), Karen Schuchardt (PNNL, working on bound uncertainty in subsurface models under SciDAC), and Vinod Tipperaju (ORNL, a leader in the development and application of Global Arrays and other PGAS programming models).

We have begun active prototyping on the ParVis data analysis project and the subsurface modeling project.

We collaborate with the NSF-sponsored CIM-EARTH project on energy-economics models.

10.4 Technologies/Techniques

We have developed the "JETS" middleware technology to enable many-task execution of MPI applications on the BG/P architecture. (See §5.2.1.)

10.5 Inventions/Patent Applications

None.

10.6 Other products

None.