

Falkon: A Proposal for Project Globus Incubation

1. A proposed name for the project

Falkon: a Fast and Light-weight task executiON framework

2. The prefix to use for the email lists that will be set up(*-dev, *-user, etc)

falkon-dev, falkon-user, etc

3. A proposed project chair, with contact information

Ioan Raicu <iraicu@cs.uchicago.edu>

4. A list of the proposed committers for the project

Ioan Raicu¹ and Yong Zhao¹ and Catalin L. Dumitrescu¹ and Ian Foster^{1,2} and Michael Wilde^{1,2}

<p>¹ Computer Science Dept., University of Chicago 1100 E. 58th Street, Ryerson Hall Chicago, IL 60637, USA <i>{iraicu,yongzh,cldumitr,foster,wilde}@cs.uchicago.edu</i></p>
<p>² Mathematics and Computer Science Department, Argonne National Laboratory Cass Ave, Chicago, IL 61637, USA <i>{foster,wilde}@mcs.anl.gov</i></p>

5. An Overview of the Aims of the Project

Many interesting computations can be expressed conveniently as data-driven task graphs, in which individual tasks wait for input to be available, perform computation, and produce output. Systems such as DAGMan, Karajan, Swift, and VDS support this model. These systems have all been used to encode and execute thousands of individual tasks. However, dispatching such tasks directly to batch schedulers has two disadvantages. First, because a typical batch scheduler provides rich functionality (e.g., multiple queues, flexible task dispatch policies, accounting, per-task resource limits), the time required to dispatch a task can be large—30 secs or more—and the aggregate throughput relatively low (perhaps two tasks/sec). Second, while batch schedulers may support different queues and policies, the policies implemented in a particular instantiation may not be optimized for many tasks.

To enable the rapid execution of many tasks on compute clusters, we have developed Falkon, a Fast and Light-weight task executiON framework. Falkon uses (1) multi-level scheduling to separate resource acquisition (via, e.g., requests to batch schedulers) from task dispatch, and (2) a streamlined dispatcher. Multi-level scheduling, introduced in operating systems research in the 1990s, has been applied to clusters by the Condor team and others, while streamlined dispatchers are found in, e.g., BOINC. Falkon's integration of the techniques delivers performance not provided by any other system [1].

5.1. Current Status

We have already designed a first architecture and prototyped on TeraGRID. Our performance results for both microbenchmarks show that Falcon throughput is around 440 tasks/sec and its scalability around 54,000 executors and 2,000,000 tasks processed in just over two hours, which are one to two orders of magnitude better than other schedulers. Applications by means of the Swift parallel programming system reduce end-to-end run time of up to 90% for large-scale astronomy and medical applications, relative to versions that execute tasks via separate scheduler submissions.

5.2. Future Work and Directions

We plan to implement and evaluate enhancements, such as task prefetching, alternative technologies, data management, and three-tier architecture [2].

- **Pre-fetching:** As is commonly done in manager worker systems, executors can request new tasks before they complete execution of old tasks, thus overlapping communication and execution.
- **Technologies:** Performance depends critically on the behavior of our task dispatch mechanisms; the number of messages needed to interact between the various components of the system; and the hardware, programming language, and compiler used. We implemented Falcon in Java and use the Sun JDK 1.4.2 to compile and run Falcon. We use the GT4 Java WS-Core to handle Web Services communications. One potential optimization is to rewrite Falcon in C/C++, (using, for example, the Globus Toolkit C WS-Core). Another is to change internal communications between components to a custom TCP-based protocol. However, dispatch rates are adequate for applications studied to date, and the primary obstacle to scaling is likely to be data access, not task dispatch.
- **Data management:** Many Swift applications read and write large amounts of data. Applications typically access data from a shared data repository (e.g., NFS, GPFS, GridFTP, web server). Thus, data access can become a bottleneck as applications scale. We expect that data caching, proactive data replication, and data aware scheduling can offer significant performance improvements for applications that have locality in their data access patterns. We plan to implement data caching mechanisms in Falcon executors, which would allow executors to populate local caches with data the corresponding task would require. In conjunction with data caching we may wish to implement a data-aware dispatcher. We will evaluate to what extent data aware dispatching reduces performance. A user can choose which dispatcher and executor to use for a specific application.
- **3-Tier Architecture:** Falcon currently requires that the dispatcher and client can each send messages to the other. Thus, each must have at least one port open in their firewall. We have implemented a polling mechanism to bypass firewalls on executors or clients, but we loose this performance and scalability port open in the firewall on which it will accept WS due to the polling mechanism vs. the notification mechanisms. Note that the dispatcher is still required to receive messages from clients and executors. Falcon also currently assumes that executors operate in a public IP space, so that the dispatcher can communicate with them directly. If (as is sometimes the case) a cluster is configured with a private IP space, to which only a head node has access, the Falcon dispatcher must run on that head node.

6. An Overview of Any Current User Base or User Community

- Swift Team (University of Chicago / USA): around 10 researchers from University of Chicago, ISI and Indiana University
- ServMark Team (Globus Incubator Project): more than 15 people from more than 4 universities
- TeraGRID: partially deployed

7. An overview of how the Candidate relates to other parts of Globus

First, Falkon is implemented as a WSRF service and a fast scheduling interface for large bags of jobs in a Grid.

Second, users of Grid workflow engines, such as Swift, Karajan, BPEL or Condor-G, will have now the choice to reduce system overloads when submitting large amounts at once of jobs into a Grid [3]. When using Swift and Falkon together, we demonstrated reductions in end-to-end run time by as much as 90% for applications from the astronomy and medical fields, when compared to the same applications run over batch schedulers [2].

Third, Falkon represents a complement for the work of Bresnahan et al. that targets a multi-level scheduling architecture specialized for the dynamic allocation of compute cluster bandwidth. A modified Globus GridFTP server varies the number of GridFTP data movers as server load changes.

8. A summary of Why the Candidate would Enhance and Benefit Globus

The schedulers used to manage parallel computing clusters are not typically configured to enable easy configuration of application-specific scheduling policies. In addition, their sophisticated scheduling algorithms and feature-rich code base can result in significant overhead when executing many short tasks. Falkon is designed to enable the efficient dispatch and execution of many small tasks, as required in Grid environments and for Grid workflow engines. To this end, it uses a multi-level scheduling strategy to enable separate treatment of resource allocation (via conventional schedulers) and task dispatch (via a streamlined, minimal-functionality dispatcher).

9. References

- [1] Ioan Raicu, Catalin L. Dumitrescu, Ian Foster, *Dynamic Resource Provisioning in Grid Environments*, TeraGRID Conference, Madison, WI, 2007 (poster).
- [2] Ioan Raicu, Yong Zhao, Catalin L. Dumitrescu, Ian Foster, Michael Wilde, *Falkon: a Fast and Light-weight task executiON framework*, submitted to Super-Computing 2007.
- [3] Catalin Dumitrescu, Ioan Raicu, Ian Foster, *Usage SLA based Scheduling in Grids*, Journal of Concurrency and Computation: Practice and Experience, Special Issue (GCC'05).