Data Diffusion Delivers Dynamic Digging

Ioan Raicu[*], Yong Zhao[*], Ian Foster[#*+], Alex Szalay[-]

{iraicu,yongzh }@cs.uchicago.edu, foster@mcs.anl.gov, szalay@jhu.edu

[*] *Department of Computer Science, University of Chicago, IL, USA*

[+] *Computation Institute, University of Chicago & Argonne National Laboratory, USA*

[#] *Math & Computer Science Division, Argonne National Laboratory, Argonne IL, USA*

[-] *Department of Physics and Astronomy, The Johns Hopkins University, Baltimore MD, USA*

1 Overview

We want to support interactive analysis ("digging") of large quantities of data, a requirement that arises, for example, in many scientific disciplines. Such analyses require turnaround measured in minutes or seconds. Achieving this performance can demand hundreds of computers to process what may be many terabytes of data. As the applications scale, data sets grow, and resources used increase, the importance of data locality will be crucial to the successful and efficient use of large scale distributed systems for many scientific and data-intensive applications. [9]

One approach to delivering such performance, adopted, for example, by Google [5, 13], is to build large compute-storage farms dedicated to storing data and responding to user requests for processing. However, such approaches can be expensive (in terms of idle resources) if load varies significantly over the two dimensions of time and/or the data of interest.

Grid infrastructures allow for an alternative *data diffusion* approach, in which the resources required for data analysis are acquired dynamically, in response to demand. As request rates increase, more resources are acquired, either "locally," if available, or "remotely" if not; the location only matters in terms of associated cost tradeoffs. Both data and applications can diffuse from low-cost archival or slower disk storage to newly acquired resources for processing. Acquired resources (computer and associated storage, and also data) can be "cached" for some time, thus allowing more rapid responses to subsequent requests. If demand drops, resources can be released, allowing for their use for other purposes. In principle, this approach can provide the benefits of dedicated hardware without the associated high costs—depending crucially, of course, on the nature of application workloads and the performance characteristics of the grid infrastructures on which the system is deployed.

This data diffusion concept is reminiscent of cooperative Web caching [22] and peer-to-peer storage systems [20]. (Other data-aware scheduling approaches tend to assume static resources [1, 2, 3, 4, 10].) Also, others have performed simulations on data-aware scheduling [12]. But the devil is in the details, and the details are different in many respects. We need to allocate not just storage but also computing power. Users do not want to retrieve the data, but to analyze it. Datasets may be terabytes in size. Further complicating the situation is our limited knowledge of workloads.

In order to explore this concept and gain experience with its practical realization, we are developing a Fast and Light-weight tasK executiON framework (Falkon) [6, 14], which provides for dynamic acquisition and release of resources ("workers") and the dispatch of analysis tasks to those workers. We describe here how Falkon has been extended to include data caching, thus enabling the data management of tens of millions of files spanning potentially hundreds to thousands of multiple storage resources. This paper presents the design of the data caching extensions to Falkon and some preliminary performance results.

2 Falkon Architecture

To enable the rapid execution of many tasks on distributed resources, Falkon combines (1) multi-level scheduling [15, 16] to separate resource acquisition (i.e. requests to batch schedulers) from task dispatch, and (2) a streamlined dispatcher to achieve one to two orders of magnitude higher throughput (440 tasks/sec) and scalability (54K executors, 2M queued tasks) than other resource managers [6].

The Falkon architecture comprises a set of (dynamically allocated) *executors*, that cache and analyze data; a *dynamic resource provisioner* that manages the creation and deletion of executors; and a *dispatcher*, that dispatches each incoming task to an executor. The provisioner uses tunable allocation and de-allocation policies to provision resources adaptively. Microbenchmarks show that DRP can allocate resources in 10s of seconds across multiple Grid sites and reduce average queue wait times by up to 95% (effectively yielding queue wait times within 3% of ideal) [14].

Prior to the work presented in this paper, we have assumed that each task scheduled by Falkon accessed its input and output at remote locations. This strategy works surprisingly well in many cases, but of course cannot scale to more data-intensive applications. Example of applications in which problems arise include image stacking [7, 8] and mosaic services [17, 18] in astronomy, which access digital image datasets that are typically large (multiple terabytes) and contain many objects (100M+) stored into many files (1M+).

Finally, Falkon support in the Swift parallel programming system [19] allows Swift applications to run over Falkon without modification.

3 Adding Data Diffusion

We introduce data diffusion by incorporating data caches in executors and data location-aware task scheduling algorithms in the dispatcher. In our initial implementation, individual executors manage their own cache content, using local cache eviction policies, and communicate changes to cache content to the dispatcher. The dispatcher can then associate with each task sent to an executor information about where to find non-

cached data. In the future, we will also analyze and (if analysis results are encouraging) experiment with alternative approaches, such as decentralized indices and centralized management of cache content.

We assume that data are never modified after initial creation. Hence, we need not check if cached data is stale. This assumption is valid for Swift applications with which we work. We have implemented four well-known cache eviction policies [21]: *Random*, *FIFO* (First In First Out), *LRU* (Least Recently Used), and *LFU* (Least Frequently Used).

Data caching at executors implies the need for data-aware scheduling. We implement three policies: *first-available*, *max-cache-hit*, and *max-compute-util*.

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

The **max-cache-hit** policy uses information about data location to dispatch each task to the executor with the largest number of data needed by that task. If that executor is busy, task dispatch is delayed until the executor becomes available. This strategy can be expected to reduce data movement operations relative to first-available, but may lead to load imbalances, especially if data popularity is not uniform.

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

In all three cases, we pass with each task information about remote resources (workers or storage systems) on which missing data can be found. Thus, the executor can fetch needed data without further lookups.

4 Performance Evaluation

We have a complete reference implementation of the data diffusion mechanisms but have yet to measure its performance. Thus, we report just some preliminary studies aimed solely at evaluating the dispatcher's ability to handle lookups of data location information. In these experiments, we assume a dataset of 1.5M files (each being 6MB) and a set of task requests each of which requires F of these files, selected randomly according to a uniform distribution (this is representative of the SDSS astronomy dataset [11]). Each of E executors has 100GB of local storage (this is representative of the TeraGrid [23] local storage resources on each compute node). We run the Falkon dispatcher using the *max-compute-util* policy, with local threads emulating the task receipt and cache update operations performed by workers; no actual computation or data transfer is performed.

Figure 1 shows measured throughputs. We see that the number of files per task that the dispatcher can handle while maintaining at least 100 tasks/sec range from around 512 (with a single storage resource) to 64 (for 32K storage resources). These results are encouraging in terms of achievable dispatcher rates.

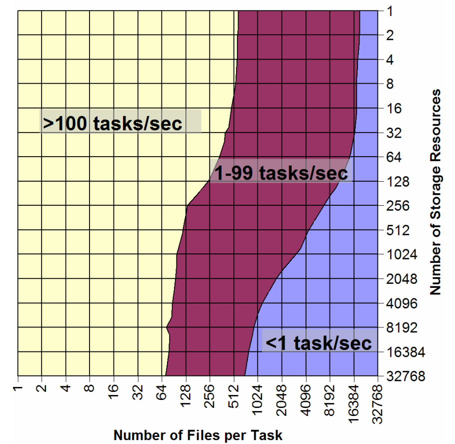

Figure 1: Data-aware scheduler performance

We need to perform more experiments on the max-cache-hit scheduling policy, as well as on the various cache eviction policies. Running micro benchmarks to get baseline measurements to how effective the proposed extensions can be in terms of utilizing local caches need to be performed. We also need to run some large scale experiments using Falkon with the data caching and new scheduling policies on real applications; we expect that the astronomy applications which we have been working with (image stacking and mosaic services) should have faster end-to-end completion time, as well as better scalability. At this point, it is hard to quantify the application performance improvements without at least having some microbenchmarks results on data cache hit rates for the data access patterns these applications exhibit, or at least to have some small scale application runs so we can extrapolate the expected performance gains.

5 Conclusions

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

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

As a first step towards exploring this concept, we have extended our Falkon system to support basic data diffusion mechanisms. Next on our agenda is exploring the performance of these mechanisms with real applications and workloads. Then, we will likely start to explore more sophisticated algorithms. For example, what should we do when an executor is released? Should we simply discard cached data, should it be moved to another active executor, or should it be moved to some permanent and shared storage resource? The answer will presumably depend on workload and system characteristics.

6 References

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

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

[3] M. Ernst, P. Fuhrmann, M. Gasthuber, T. Mkrtchyan, C. Waldman. "dCache, a distributed data storage caching system," Chep 2001.

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

[5] S. Ghemawat, H. Gobioff, S.T. Leung. "The Google file system," Procedings of the 19th ACM SOSP (Dec. 2003), pp. 29-43.

[6] I. Raicu, Y. Zhao, C. Dumitrescu, I. Foster, M. Wilde. "Falkon: a Fast and Light-weight tasK executiON framework", under review at IEEE/ACM SC 2007.

[7] I. Raicu, I. Foster, A. Szalay. "Harnessing Grid Resources to Enable the Dynamic Analysis of Large Astronomy Datasets", IEEE/ACM SC 2006.

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

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

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

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

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

[13] J. Dean, S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters, OSDI'04: Sixth Symposium on Operating System Design and Implementation, San Francisco, CA, December, 2004.

[14] I. Raicu, C. Dumitrescu, I. Foster. Dynamic Resource Provisioning in Grid Environments, to appear at TeraGrid Conference 2007.

[15] G. Banga, P. Druschel, J.C. Mogul. "Resource Containers: A New Facility for Resource Management in Server Systems." Symposium on Operating Systems Design and Implementation, 1999.

[16] J.A. Stankovic, K. Ramamritham,, D. Niehaus, M. Humphrey, G. Wallace, "The Spring System: Integrated Support for Complex Real-Time Systems", Real-Time Systems, May 1999, Vol 16, No. 2/3, pp. 97-125.

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

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

[19] 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", to appear at IEEE Workshop on Scientific Workflows 2007.

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

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

[22] R. Lancellotti, M. Colajanni, B. Ciciani, "A Scalable Architecture for Cooperative Web Caching", Proc. of Workshop in Web Engineering, Networking 2002, Pisa, May 2002.

[23] TeraGrid, http://www.teragrid.org/