Tashi: Location-aware Cluster Management

Michael A. Kozuch, Michael P. Ryan, Richard Gass, Steven W. Schlosser, David O'Hallaron
Intel Research Pittsburgh
James Cipar, Elie Krevat, Michael Stroucken, Julio López, Gregory R. Ganger
Carnegie Mellon University

Abstract

Big Data applications, those that require large data corpora either for correctness or for fidelity, are becoming increasingly prevalent. Tashi is a cluster management system designed particularly for enabling cloud computing applications to operate on repositories of Big Data. These applications are extremely scalable but also have very high resource demands. A key technique for making such applications perform well is Location-Awareness. This paper demonstrates that location-aware applications can outperform those that are not location aware by factors of 3-11 and describes two general services developed for Tashi to provide location-awareness independently of the storage system.

1 Introduction

Big-data computing is perhaps the biggest innovation in computing in the last decade. [4]

Increasingly, the most interesting computer applications are those that compute on large data corpora. These "Big Data" applications draw from information sources such as web crawls, digital media collections, virtual worlds, simulation traces, and data obtained from scientific or medical instruments. Historically only of interest to a narrow segment of the computing community, Big Data applications now play a significant role in all aspects of society– from scientific study to enterprise data mining to consumer web applications.

These applications, beyond simply operating on data that is big, are also typically *data hungry* in that the quality of their results improves with the quantity of data available. Consequently, a strong need exists for computing technologies that are scalable– to accomodate the largest datasets possible.

Fortunately, these applications are typically disk bandwidth limited (rather than seek-limited) and exhibit extremely good parallelism. Therefore, commodity cluster hardware, when employed at scale, may be harnessed to support such large dataset applications. For example, the cluster at Intel Research Pittsburgh that is part of the OpenCirrus consortium (`http://opencirrus.org/`) consists of a modest 150 server nodes, yet provides more than 1000 computing cores and over 400 TB of disk storage– enough to accomodate many current Big Data problems.

One unfortunate ramification of data set size is potentially inescapable; namely, that Big Data sets are relatively immobile. With a 1 Gbps connection to the Internet, moving a 100 TB data set into or out of a cluster like the one mentioned above would require approximately 10 days. (The actual state of affairs is worse; access to the Pittsburgh cluster is through a T3 (45 Mbps) connection.) Consequently, unless the ratio of transfer bandwidth to data set size increases dramatically, computation on Big Data sets will be *in situ*.

In other words, Big Data facilities will host not only data sets, but also all the computation that operates on those data sets. A site will enable such computation through one or both of two models. In the first model, sites provide a traditional query service through a narrowly defined interface (e.g. contemporary image search). In the second model, which combines Cloud Computing with Big Data, sites provide a computation-hosting framework where users bring their own, custom applications to the facility to operate on the data. Because of the flexibility provided by the second approach, we believe that, in the future, hosted computation will play an increasingly significant role in the consumption of Big Data.

The authors are currently developing an open-source, cluster-management software package called Tashi that is designed to support Cloud Computing applications that operate on Big Data. Currently, Tashi is in production use at the OpenCirrus site mentioned above, and the project is hosted by the Apache Software Foundation incubator. A key feature of Tashi is its support for *location-aware computing*, which can impact performance by a factor of 3-11, even in modestly-sized clusters, as shown in Section 2. This paper presents two basic services, described in Section 3, that support location-aware com-

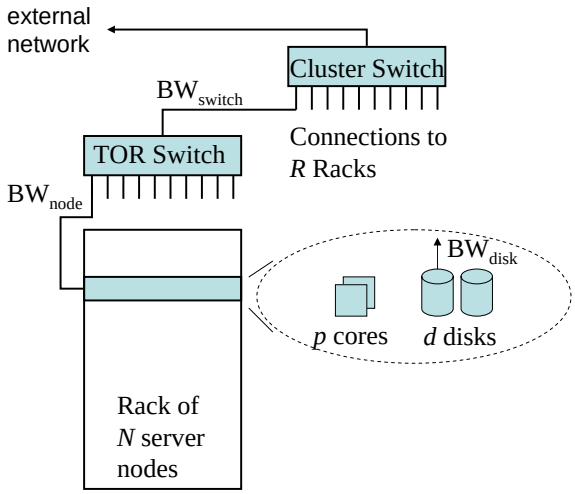

Figure 1: Example cluster organization with minimal networking. The uplinks from the Top of Rack (TOR) switches to the Cluster Switch often introduce communication bottlenecks.

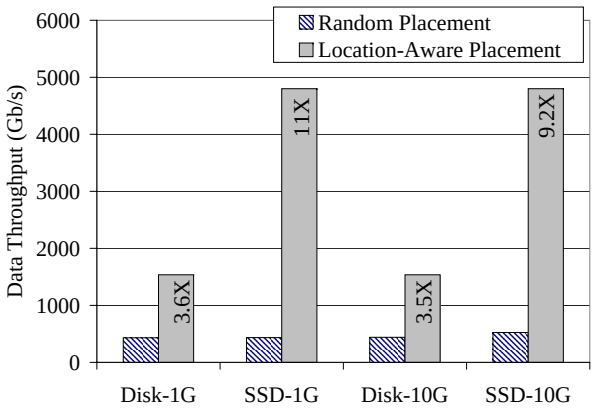

Figure 2: Performance comparison of location aware task placement with random task placement. The labels on the data bars show the performance improvement for the Location-Aware Placement relative to the Random Placement.

puting. These services, a *Data Location Service* and a *Resource Telemetry Service*, provide a standard interface between application runtimes and storage systems and are essential to optimizing the performance Big Data applications.

2 System Considerations

Figure 1 depicts the hardware organization we assume for modest-sized Big Data clusters. The server nodes are organized into R racks; each rack contains N nodes. Each node contains p processors and d disk units, which are either traditional magnetic disks or solid state drives (SSDs), and all nodes in the rack are connected to a commodity switch, the Top of Rack (TOR) switch. The uplinks of the R TOR switches are connected to a single cluster switch or router.

The Big Data repository is assumed to be distributed across the disk devices in the server nodes, as opposed to being maintained in dedicated storage. This simple arrangement provides a total of $R*N*d$ disks with a minimal investment in networking components. While more sophisticated networking arrangements can provide higher cross-sectional bandwidth (e.g. [3]), this improvement comes at the cost of a reduced number of compute nodes, given a fixed capital budget for the cluster. As an alternative, we are investigating techniques that reduce pressure on the network connectivity, which enables a greater portion of the available budget to be spent on computing resources.

In particular, systems based on the MapReduce paradigm [6, 2] attempt to schedule computing tasks to execute on the nodes that contain the data those tasks expect to consume. A straightforward analysis can show the effectiveness of this approach. In particular, when tasks consume data from the node on which they are executing, the data can be consumed at the disk bandwidth rate. When tasks are not properly placed, they pull data from a distant node across the network, and this data is now constrained not only by the bandwidth of the network components, but by any other contending data flows.

To evaluate this effect, we developed an analytical model based on $R = 20$, $N = 30$, $d = 2$, and $BW_{switch} = 10Gbps$. Figure 2 shows the difference, in terms of delivered throughput, between a *Location-Aware Placement*, where tasks consistently consume data from the node on which they execute, and a *Random Placement*, where tasks are placed randomly in the cluster without regard to data location. As shown in the figure, we experimented with two types of disks: a traditional magnetic disk with $BW_{disk} = 80MB/s$ and a solid state disk (SSD) with $BW_{disk} = 250MB/s$. We also evaluated two values for BW_{node}: $1Gbps$ and $10Gbps$. The results show clearly that an intelligent placement may improve the performance of the cluster by 3-11 times. In other words, a small cluster with location-aware placement can outperform a significantly larger (up to ten times larger) cluster with random placement.

Given that Location-Aware Placement is crucial to good performance in Big Data applications, should all Big Data applications be written in the MapReduce

style? The answer is no, for two reasons.

First, as has been described previously [9], not all applications can be efficiently implemented as MapReduce programs. Software developers who understand the data flow of their application should not be forced to use any particular tool to express their program– instead they should be able to leverage the mechanisms deemed most suitable for any particular task. Of course, programmers will want to take advantage of location-aware execution for the performance reasons outlined above. However, location-awareness is a property that can be abstracted for use in many cluster environments, not just MapReduce.

Second, in hosting environments, users with varying software needs will place a management burden on the cluster administration team in the form of software package installations. Rather than providing a single application interface to all users, a better solution is to provide virtual machine containers; with such an arrangement, users can manage their own software installations.

Given these observations, the guiding principles for the Tashi design are as follows:

1. Tashi will provide cluster management services– particularly, Tashi will host and administer virtual machine containers.
2. Where possible, Tashi will provide the mechanisms necessary for location-aware placement of computing tasks.
3. Tashi will be virtual machine monitor (VMM) neutral; the current implementation supports KVM and Xen.
4. Tashi will be distributed file-system neutral. In particular, support is planned for both centralized (e.g. NFS) and decentralized file systems (e.g. HDFS and PVFS).

3 Tashi Software Design

Tashi is a virtualization-based cluster management system that provides facilities for managing virtual machines. Users of Tashi are able to create collections of virtual machines that run on the cluster's physical resources. These virtual machine collections form "virtual clusters" that operate on the Big Data stored in the cluster, and when distinctions are necessary, users who create virtual machines in Tashi are called *virtual cluster owners*. Virtual clusters may host services that are then consumed by *clients*, users that interact with, but do not own, the virtual machines managed by Tashi.

A sample listing of the Tashi management interface is provided in Table 1. In terms of these basic virtual machine management facilities, Tashi is similar to Cluster-on-Demand [8], Usher [10], Amazon's EC2 infrastructure [1], and Eucalyptus [11]. Where Tashi differs from these systems is in its support for location-aware computing.

3.1 Filesystems

Because Big Data applications demand access to very large datasets, a cluster file system is required for any system designed to support such applications. Parallel data access is also a key property of any viable Big Data storage systems. While Tashi is designed to be interoperable with many different distributed file systems, two particularly interesting, currently available, candidates are the Hadoop file system [2, 7] and PVFS [5].

Because application developers will typically consider parallel data access as well as compute parallelism very carefully when writing Big Data applications, completely abstracting differences between file systems may not only be unnecessary, it may be counter-productive. Consequently, we believe exposing important properties of the file systems (such as if data blocks are striped across data nodes or randomly placed) is desirable, and while we believe that developing an API that is common across file systems may be desirable, we expect that most high-performance applications will access distributed file systems through their native interfaces, particularly for data access.

However, performance-sensitive applications, as a class, will benefit from exploiting data location information, and the access of this information is not necessarily performance-critical. Therefore, providing a standardized facility across file systems for accessing location information will enable applications to become location-aware with minimal programmer effort. By "location aware", we mean that, when an application determines that it needs to spawn a task to operate on data block B of file F, it is able to determine on which server node block B is physically stored so that the task could possible be instantiated on that node, or a nearby node (one in the same rack, e.g.).

In our current work, we assume that all data placement decisions have been made *a priori*. Due to the high cost of re-distributing very large datasets, most applications will need to be scheduled with the data layout predetermined. In future work, however, we plan to relax this assumption and address questions relating to how data placement and scheduling operations may work in concert.

3.2 Application Software Stacks

To take advantage of location information, Big Data applications typically rely on a location-aware runtime, which is responsible for interacting with the file sys-

Table 1: Listing of sample Tashi management commands.

createVm –name < *string* > [–cores < *n* >] [–memory < *n* >] –disks < *string* >
createMany –basename < *string* > [–cores < *n* >] [–memory < *n* >] –disks < *string* > –count < *n* >
destroyVm –instance < *n* >
destroyMany –basename < *string* >
migrateVm –instance < *n* > –targetHostId < *n* >
pauseVm –instance < *n* >
unpauseVm –instance < *n* >
suspendVm –instance < *n* > –destination < *n* >
resumeVm –name < *string* > [–cores < *n* >] [–memory < *n* >] –disks < *string* > –source < *string* >
shutdownVm –instance < *n* >
vmmSpecificCall –instance < *n* > –arg < *string* >

tem to extract location information and make task placement decisions. Such runtimes are often explicit components of the system programming model, as in the case of the MapReduce model. In some cases, though, applications include custom runtime components. Therefore, the service that provides location information must be queryable not only from well-know runtime components, but from individual applications as well.

The high-level software architecture for providing location-aware services is shown in Figure 3. We consider two different environments for location-aware applications. In part (a) of the figure, we see an application stack executing directly on the host server node. Part (b) depicts a similar application structure, except that it is executing within a virtual machine that is located on that node. In both cases, the location-aware runtime and/or application accesses two services to determine location information: a *Data Location Service*, which provides a mapping from file data blocks to storage node identifiers, and a *Resource Telemetry Service*, which provides information regarding the relative location of resources such as storage node identifiers.

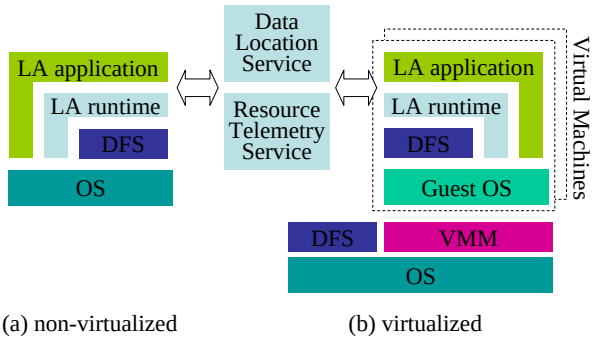

Figure 3: Software components supporting location awareness in Tashi. Location-aware (LA) applications leverage the Data Location Service and Resource Telemetry Service to obtain information regarding the location of data objects in the Distributed File System (DFS) and may execute either (a) directly on the host infrastructure or (b) inside virtual machine containers.

3.3 Exposing Location Information

The Data Location Service provides a mapping from file data blocks to storage node identifiers, which may be simple hostnames or IP addresses. In our current design, each data location service is associated with a particular file system; hence, specifying a named data location service implicitly identifies a particular file system. The data location service is a small daemon, running somewhere in the cluster, that provides the interface shown in Figure 4.

This interface assumes that the file system divides files into a series of data blocks numbered sequentially. Each block corresponds to a particular byte range of the file, but may be replicated within the file system (to provide failure tolerance, for example). The `blockInfo` struc-

```
struct blockInfo {
  encodingType type;
  byteRange range;
  list<hostId> nodeList;
};

list<blockInfo>
getBlockInfoByteRange(fileId f,
                      byteRange r);
```

Figure 4: Data Location Service (DLS) interface. DLS clients specify a range of bytes in a large file, and the service returns a list of block descriptions the application may use to reconstruct the byte range.

```
typedef double metricValue;

metricValue
getMetric(hostId from, hostId to,
          metricType t);

list< list<metricValue> >
getAllMetrics(list<hostId> fromList,
              list<hostId> toList,
              metricType t);
```

Figure 5: Resource Telemetry Service (RTS) interface. Clients are able to use getMetric() to obtain the distance from one resource to another. With getAllMetrics(), clients are able to obtain the all-pairs distances with a single call. Note that the interface does not assume symmetry; a measure from host1 to host2 might yield a value that is not equal to the same measure from host2 to host1.

ture returned by calls to `getBlockInfo()`, then, provides a listing of all nodes that maintain a copy of the given block. In fact, the replicas may not even be exact copies; the `encodingType` enables the data location service to identify blocks that are encoded with M-out-of-N codes, for example [12].

In some cases, the information returned by the data location service may be sufficient for location-aware systems to place computation tasks. However, note that two scenarios, in particular, require the additional information provided by the Resource Telemetry Service (RTS): determining if two hosts reside in the same rack and determining on which host virtual machines reside. The RTS is responsible for relating information such as notions of distance between hostId values. The API for this service is shown in Figure 5. We have intentionally left the notions of `metricType` generically specified because we believe that this interface may be useful for a number of different metrics such as network hop count, observed latency, observed bandwidth, nominal bandwidth, and other application-defined measures of distance.

However, this interface is particularly useful in identifying notions of network distance. With a simple map of the cluster installation, the resource telemetry service is able to supply useful information such as how many switches must be traversed on communication paths from one hostId to another; given our simple network topology, a value of 1 indicates that the two hosts are in the same rack, and a value of more than one indicates that communication must traverse the central switch.

Further, note that virtualization obscures some location information. If VMs are provided with cluster-wide IP addresses, determining which VM is "closest" to a particular data block (and hence should host a task operating on that block) is challenging. The responsibility of resolving such vagaries falls to the Resource Telemetry Service. For example, the switch-hop metric might be defined to return a value of 0.5 when comparing two hostIds if one is a virtual machine and the other is the host on which the virtual machine resides. Such an assignment might be used to inform the scheduling agent that placing a task directly on a host machine is preferable to placing the task in a VM running on that host.

3.4 Leveraging Location Information

In order to support applications with different levels of location awareness, Tashi aims to provide a flexible interface that separates location information, resource information and scheduling decisions in different components. Location information is consumed by the scheduling modules in Tashi and location-aware clients. For example, when executing applications that are not location aware, Tashi handles the initial allocation of VMs to hosts based on resource requirements (e.g., VM memory size) and resource availability at the hosts. VMs may be relocated in response to (a) requests submitted through the client API or (b) decisions made by the Tashi scheduling agent. In contrast, location-aware applications query the Data Location Service to obtain information about the placement of the input data. The application can then use the Resource Telemetry Service to determine the distance between the input data and the initial set of VMs. Based on this information, the application can determine a task to host assignment that reduces the overall data movement. More sophisticated location-aware applications could interact directly with the Tashi scheduling agent to request a particular placement of the VMs based on the location of the input data. A key observation, here, is that the location service architecture enables a decoupling of admission control, scheduling, and data placement.

3.5 Adapting to Location Changes

Note that while the resource telemetry service provides the right abstraction for determining appropriate placement decisions in the presence of virtualization, they do not currently provide interfaces that enable adaptation in the presence of all location changes. Consider two cases: location-aware VM placement and location-aware task placement.

In the first case, the cluster manager explicitly places virtual machines according to some assumptions regarding the data access patterns of those virtual machines. If a higher-priority task arrives, the cluster manager may

consolidate/migrate those virtual machines– yielding a sub-optimal placement. As the cluster manager set up both the initial and subsequent schedules, it has the information necessary to inform the re-scheduled virtual machines of their new status in order to make any necessary adjustments.

In the second case, an application distributed across a number of virtual machines may have optimized the data access patterns of the individual tasks *after* the VMs were placed. If the cluster manager subsequently redistributes the VMs across the cluster, the application may remain oblivious to the change and continue executing with, what is now, a sub-optimal task placement. To compensate, tasks could either register callbacks with the cluster manager or poll the Resource Telemetry Service to determine if there were any changes in their status. Determining the reasonable interface for change notification is part of our current work.

4 Current Status

The Tashi project is currently in production use at the OpenCirrus cluster at Intel Research Pittsburgh, a cluster of approximately 150 server nodes, comprising 1000 cores and 400 disks.

The source code is available at the Apache Software Foundation incubator where it is hosted. The current implementation is under active development and should be considered of alpha quality. In particular, we have prototyped the Data Location and Resource Telemetry Services but those services are not yet part of the mainline implementation. Initial results are promising however; a test application was able to leverage the prototype services to significantly improve its performance. The standalone application read through a 1 TB dataset stored in HDFS on a 28-node cluster using a random task layout in 139 minutes, but using a location-aware assignment, the same application read the same dataset in 14.5 minutes–nearly a 10X improvement.

5 Conclusions

Location-awareness is an important tool in providing Big Data services with good performance; even on modest clusters the performance impact can be significant. We propose that the Data Location and Resource Telemetry Services offer the right abstraction for general, cluster-wide location services.

References

[1] Amazon elastic compute cloud. URL http://aws.amazon.com/ec2/.

[2] Hadoop core. URL http://hadoop.apache.org/core/.

[3] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *SIGCOMM '08: Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, pages 63–74, New York, NY, USA, 2008. ACM.

[4] R. E. Bryant, R. H. Katz, and E. D. Lazowska. Big-data computing: Creating revolutionary breakthroughs in commerce, science, and society. In *Computing Research Initiatives for the 21st Century*. URL http://www.cra.org/ccc/initiatives.

[5] P. H. Carns, W. B. Ligon, III, R. B. Ross, and R. Thakur. Pvfs: a parallel file system for linux clusters. In *ALS'00: Proceedings of the 4th annual Linux Showcase & Conference*, pages 28–28, Berkeley, CA, USA, 2000. USENIX Association.

[6] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. In *OSDI'04: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation*, pages 10–10, Berkeley, CA, USA, 2004. USENIX Association.

[7] S. Ghemawat, H. Gobioff, and S.-T. Leung. The google file system. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 29–43, New York, NY, USA, 2003. ACM.

[8] D. Irwin, J. Chase, L. Grit, A. Yumerefendi, D. Becker, and K. G. Yocum. Sharing networked resources with brokered leases. In *ATEC '06: Proceedings of the annual conference on USENIX '06 Annual Technical Conference*, pages 18–18, Berkeley, CA, USA, 2006. USENIX Association.

[9] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *EuroSys '07: Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, pages 59–72, New York, NY, USA, 2007. ACM.

[10] M. McNett, D. Gupta, A. Vahdat, and G. M. Voelker. Usher: an extensible framework for managing custers of virtual machines. In *LISA'07: Proceedings of the 21st conference on Large Installation System Administration Conference*, pages 1–15, Berkeley, CA, USA, 2007. USENIX Association. ISBN 978-1-59327-152-7.

[11] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. he eucalyptus open-source cloud-computing system. In *Cloud Computing and its Applications (CCA'08)*, 2008. URL http://cca08.org.

[12] J. J. Wylie, M. W. Bigrigg, J. D. Strunk, G. R. Ganger, H. Kiliççöte, and P. K. Khosla. Survivable information storage systems. *Computer*, 33(8):61–68, 2000.