The Need to Consider Hardware Selection when Designing Big Data Applications Supported by Metadata

Nathan Regola
Department of Computer Science and Engineering and
Interdisciplinary Center for Network Science and Applications
University of Notre Dame
USA

David A. Cieslak
Aunalytics, Inc.
Notre Dame, Indiana
USA

Nitesh V. Chawla
Department of Computer Science and Engineering and
Interdisciplinary Center for Network Science and Applications
University of Notre Dame
USA

ABSTRACT

The selection of hardware to support big data systems is complex. Even defining the term "big data" is difficult. "Big data" can mean a large volume of data in a database, a MapReduce cluster that processes data, analytics and reporting applications that must access large datasets to operate, algorithms that can effectively operate on large datasets, or even basic scripts that produce a needed resulted by leveraging data. Big data systems can be composed of many component systems. For these reasons, it appears difficult to create a universal, representative benchmark that approximates a "big data" workload. Along with the trend to utilize large datasets and sophisticated tools to analyze data, the trend of cloud computing has emerged as an effective method of leasing compute time. This chapter explores some of the issues at the intersection of virtualized computing (since cloud computing often uses virtual machines), metadata stores and big data. Metadata is important because it enables many applications and users to access datasets and effectively use them without relying on extensive knowledge from humans about the data.

INTRODUCTION

Big data systems have emerged as a set of hardware and software solutions to allow organizations to obtain value from the increasing volume and complexity of data that is captured. Web sites are one example of leveraging big data systems and data to improve key business metrics. For example, large organizations often have a portal site that is a vital part of their business operations, whether it a private extranet, a public site for news, or an e-commerce site. While major public portal sites often appear that they are one seamless site, they are often built from many separate applications. For example, a news site may have an application that lists the top ten most popular news articles or an ecommerce site may recommend products for a user. Increasingly, these applications or components are driven by big data systems. Major portal sites are essentially large distributed systems that appear as a seamless site. This approach allows the operators of the site to experiment with new applications and deploy new applications without impacting existing functionality. It also importantly enables the workflow of each application to be distinct and supported by the necessary hardware and application level redundancy that is appropriate for that specific application. An e-commerce site would likely invest substantial resources to ensure that the "shopping cart" application was available with an uptime of 100%, while the "recommended products" application may have a target uptime of 99.9%. Likewise, a news site may decide that the front page should be accessible with a target uptime of 100%, even during periods of extreme load such as on a national election day. The news site may decide that the "most popular articles" application should have a target uptime of 99.99%. For example, a small pool of servers may present the content for display that is produced by each application, but each application may have its own internal database servers, Hadoop cluster, or caching layer.

News Site Use Case

Assume that a news site began their "most popular articles application" by ranking the frequency of news article displays. The intent of the "most popular articles application" on a news site is often to increase the length of a site visit and this is naturally linked to the business objectives of an organization. If the news site is revenue driven then increasing the length of a site visit increases the likelihood that the user will click on advertisements that generate revenue for the owner. Given this background, assume that in the initial version of the "most popular articles" application, users in the southern United States caused the most popular article to be an article that discussed an impending hurricane that might hit the southern United States. Users in the central United States may only rarely read this "most popular article" since it is not relevant to users in the central United States. Calculating the most popular articles on a news site is relatively easy, assuming that a well designed database of activity exists. However, this simple calculation isn't necessarily optimal from a business metric standpoint. The user should be presented the "most popular articles" that are actually relevant or are likely to be clicked on, since the goal is to increase traffic, not just present a ranking of the most read articles to the user. Perhaps the data scientist discovers that weather articles are generally sought out by users that are aware of impending weather events or are rarely read outside of the immediate geographic area of the weather event. The data scientist may decide to utilize information concerning the type of the article and the geographic area discussed by an article to determine whether it should be present in a "most popular articles list." Both enhancements require more complex data and a more complex query. For example, in the simplest revision of the "most popular articles" application, articles must be tagged with the type of article: "weather" or "non-weather" and the geographic area discussed in the article. These tags can likely be created by the article author. When a user visits the site, each user's geographic location must be approximated so that they can be mapped to a "geographic area" that the news site recognizes. The type of article and the geographic area discussed in the article also need to be retrieved, and the algorithm can determine which articles will be the "most popular articles" for that user, considering the type of each article and the geographic area of the user and article.

The article's tags (metadata) and any information that is utilized to approximate the location should be available for rapid retrieval. This requirement is often imposed because the "most popular articles" application will need to calculate the result quickly so that the web page can be generated and displayed within a fraction of a second. An operational data store may be utilized to store and recall the data that is needed for the application. An operational data store could be implemented as an in-memory database to provide low latency responses to the "top articles" application. The hardware to support the in-memory database may be a machine with a high speed bus connecting the CPU, memory, and network card.

The results of user behavior (e.g. user X read article Y or user X viewed advertisement Z) may be captured in the data warehouse for longer term analysis to develop improved algorithms, measure advertisement revenue, and monitor site traffic over time. The hardware to support a data warehouse is often a large volume of storage and compute capacity. Storage can be implemented as a storage area network, Hadoop file system, or directly attached local storage.

The site and its applications will likely change over time. For example, the "top articles application" may have begun with five articles for all users but is later changed to display ten articles to all users. A metadata store could track the number of articles that the top articles application displayed during a given date range. Analytics and reporting applications would utilize the metadata store and the data warehouse to explore the relationship between the number of articles displayed through the "top articles application" and the number of articles that users read, including information about their geographic area.

BACKGROUND

The performance of the application is often the ultimate concern of the big data system designer. Business managers are not concerned with which components are utilized, but merely that the system will achieve its performance targets such as precision, recall, and "response to application within .1 seconds." Therefore, the system designer's job is to determine which components and hardware can reach the desired performance targets when coupled with an appropriate algorithm. Obviously, the capabilities of current hardware technologies are useful in order to determine the latency and throughput that is achievable for a given budget. For example, system designers could implement an operational data store to create near real-time response rates. However, an operational data store would be cost prohibitive for an archived dataset where all of the data did not have to be close to the applications as memory and solid state (or flash) storage has traditionally been more costly per gigabyte than a magnetic disk. Ultimately, the applications that will execute on the hardware need to be considered as part of the decision to ensure that the application will be able to meet its performance targets when implementation is complete as various implementations of algorithms will result in differing performance results.

Background in Virtualization

The use of a virtualized environment (such as most cloud providers) presents additional complexity to the big data system architect, due to the fact that virtualization adds an additional source of overhead to the system that must be factored into the hardware and software selection process. Virtualization also enables fault tolerance when the hardware is experiencing a failure (such as degraded network connectivity because one network card in a team has failed or a disk has failed in a RAID array) and the operator wishes to move the workload to another piece of hardware while the repairs are occurring. Obviously, many modern applications that are intended to execute in distributed environments have some support for fault-tolerance, but some applications are not designed to handle hardware failures. Alternatively, it may be more efficient to relocate a workload rather than restart the execution of the workload (MPI jobs without check-pointing are one situation where the ability to migrate a workload from one failing node to a new node may save significant computational time over restarting a job). Virtualization has been utilized for this type of fault-tolerance (when hardware errors are detected on a node) with MPI jobs without MPI checkpointing (Nagarajan, Mueller, Engelmann, & Scott, 2007).

Types of Virtualization Overhead

The type of overheard present in a virtual machine can be classified into two main categories for purposes of discussion--CPU virtualization overheard and I/O virtualization overhead. I/O virtualization has an overheard that is above 0% (an ideal virtualized environment would be 100% efficient, thus having an overhead of 0%) and may affect the efficiency of a cluster to the point that adding additional nodes does not speedup the application. CPU virtualization overhead will affect the runtime of applications, especially for applications that result in a workload that is memory access intensive. These two types of overheads are useful to think about in the context of building large systems that span multiple nodes. A single node, N_1 will be efficient at task T if its CPU efficiency is high (assuming that the task is not bottlenecked by I/O constraints). Expanding beyond one node, high I/O efficiency allows the task T to scale efficiently across multiple nodes (assuming that the application is designed to be distributed across multiple nodes).

To determine the CPU efficiency, it may be prudent to benchmark a single virtualized node and compare the performance to performance on the same operating system on the same hardware. This will allow the benchmarking process to establish the CPU overhead on a given node and rule out issues resulting from significant overhead from a workload with specific patterns of memory accesses that stress the page table mapping strategies. For example, the hypervisor may not take advantage of the Intel EPT (Extended Page Tables) or AMD RVI (Rapid Virtualization Indexing) capability or the capability may not be enabled in the BIOS. VMWare ESX implemented support for these technologies in version 3.5 and demonstrated performance increases of up to 42% for benchmarks that stress the memory management unit (Bhatia, 2009).

One strategy to determine if there are any CPU or memory management bottlenecks in the virtual machine is to run the target application (or benchmark) on one machine, first utilizing one core, and scaling up the application to N cores, where N is the number of cores in the server on the native hardware, and then in the virtualized operating system. The only caveat to this approach (discussed in more depth below) is to ensure that there is not significant unintended I/O activity.

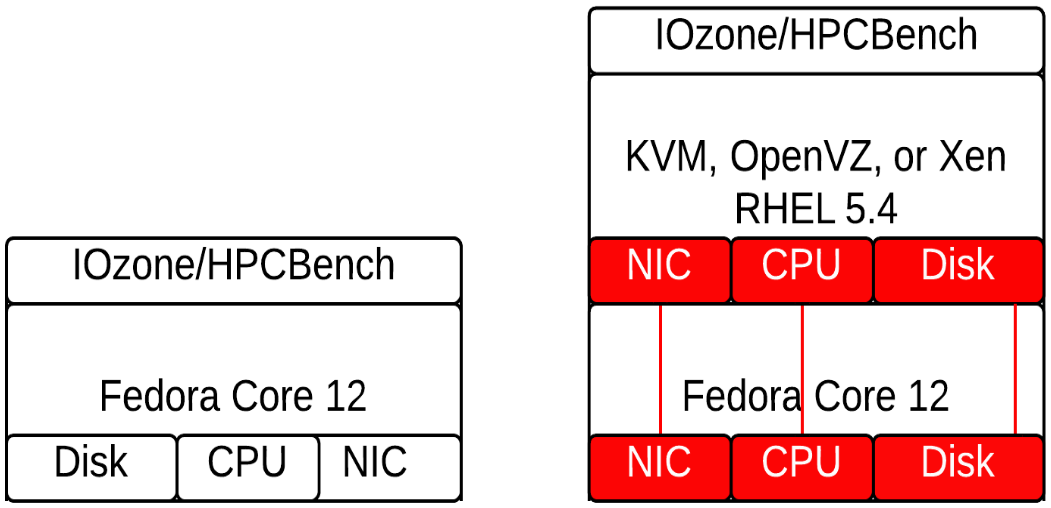

Figure 1. Diagram showing data path when benchmarking with an operating system running on native hardware compared to benchmarking with a guest operating system and a host operating system (the lines show the data path to illustrate the longer data path in a virtualized system where additional overhead can be introduced).

CPU virtualization overhead can be approximated by running a given application in a virtual machine and running the same application on the same native hardware that powered the virtual machine. The best method of setting up the experiment is not quite as easy as it may appear. The same conditions should exist when the application is executed in the virtual machine and when the application is executing in the native operating system. Establishing these “conditions” is perhaps largely a problem of defining the comparison. For example, should the virtual machine be running so that the virtualized operating system kernel is consuming some RAM during the test on native hardware since the underlying operating system and the virtualized operating system were running during the benchmark of the virtualized operating system? Should this virtualized operating system also have an application running to ensure that some memory is being actively used to force the hypervisor to deal with frequent memory accesses and page table mappings from both the virtual machine and the “native” operating system? Whether an experimenter should create an experiment that allows the virtual machine to run while the benchmark application is executing on the native hardware (since utilizing the native operating system for a task does not strictly require a virtual machine to be running since the virtual machine does not contribute to the work being completed in the native operating system, while a benchmark executing in the virtualized operating system requires both the hypervisor or native operating system and the virtual operating system to be executing) is perhaps a question of semantics and the goal of the experiment.

The major CPU manufacturers have been introducing additional hardware support to improve the performance of memory accesses in virtualized (or guest) operating systems for some time. A 2010 study found that the maximum CPU overhead when executing the NASA Parallel Benchmark Suite was approximately 5% (Regola & Ducom, 2010). The loss in efficiency may be palpable to consumers of cloud resources if they intend to accomplish a given task within a required elapsed time. To compensate for the loss of efficiency (if there are more strict timing requirements that require higher efficiency in the virtualized operating system), a consumer could lease more CPU capacity (either by leasing a more powerful processor, or if that is not possible, an additional machine). The efficiency of CPU virtualization will likely increase as CPU manufacturers incorporate additional hardware features to support virtualization.

Hardware enhancements to support efficient CPU virtualization enabled lower cost cloud computing and allow more servers to be consolidated on a single physical server. Low I/O efficiency is problematic because if I/O overhead is too high, it can drastically reduce the size of a cluster that can be efficiently operated in a virtualized environment. For example, a 32 core MPI cluster (each node had 8 cores) with a one gigabit Ethernet connection resulted in an average speedup across various “benchmark kernels” (workloads) in the NASA Parallel Benchmark Suite of 1.01 when the ideal speedup would be 4 (Regola & Ducom, 2010). However, an InfiniBand connection (20Gbps and lower latency) resulted in an average speedup of 3.05 when the ideal speedup would be 4 (Regola & Ducom, 2010). This demonstrates the problem introduced above; suboptimal network performance can limit the ability to efficiently add nodes to a cluster of machines and this is exacerbated in a virtualized environment where the I/O overhead can be significant.

Relationship of Big Data to I/O Virtualization Efficiency

I/O virtualization is problematic for big data problems because big data applications often require a cluster of nodes that communicate in some manner. When dealing with big data problems, typically a large volume of data is transferred through network connections (whether intentionally or as a result of requests to local storage that is actually not “local” to the node and is funneled through network connections). Even in paradigms such as MapReduce that attempts to keep data close to the processors that will ultimately analyze the segment of data stored “locally”, the data must be loaded to the nodes, and data is often replicated to other nodes in the cluster for fault tolerance, even if it is locally accessible for analysis. This point demonstrates that is often important to be concerned with other operations that occur in distributed systems, such as data replication, when using virtual machines since the performance of various steps could drastically alter the performance of the overall system depending on the frequency of data loading, replication, backups, etc.

Virtualized environments, such as Amazon’s EC2 offering, typically utilize complex storage systems that are not physically a part of the servers containing the virtualized nodes. These storage systems are often accessed through network connections. Depending on the design of the cloud environment and choices of the user, the network bandwidth available for MapReduce “application” could be shared with the network bandwidth that is also being used for the “local disk” since the storage blocks may actually be located on a remote system. Often, major cloud providers place “local storage” disk blocks on remote systems for scalability, performance and redundancy. Even though the MapReduce/Hadoop node (this describes the situation where a user builds their own MapReduce/Hadoop node and does not utilize the Amazon Hadoop service) believes that that disk is local, the disk blocks are actually stored on a remote system and disk requests generate network traffic when data is needed. This conflicts with the design assumptions of Hadoop and MapReduce that a node’s data is actually stored locally.

In the case of allowing the Hadoop file system to store its data on an EBS volume, MapReduce is actually accessing the “local” data through a network request. One metric that is often used to describe I/O performance (especially in relation to solid state disks) is IOPS, or input/output operations per second. For example, Amazon allows users to provision a standard EBS volume (EBS volumes appear as “local disks” in Amazon’s EC2 product) with a target IOPS of 100 or an enhanced volume with a user specified IOPS rate up to a maximum of 2000 IOPS per EC2 instance (Amazon Web Services, Inc.). The enhanced EBS volume includes dedicated bandwidth for communication to the EBS volume that is distinct from the bandwidth that is allocated to the customer’s operating system’s network connection. Amazon also supports multiple EBS volumes per instance, but the underlying bandwidth dedicated to EBS volumes ranges from 500Mbps to 1000Mbps depending on the instance type. Larger instances, such as the cluster compute nodes, utilize a 10Gbps network connection that appears to be shared with network requests that are directed to EBS volumes (Amazon Web Services, Inc.).

I/O is not always 100% efficient and in some cases the virtualization layer inhibits scalability. Fortunately, ongoing improvements are occurring in virtualized I/O performance with the introduction of technology such as PCI-SIG Single Root I/O Virtualization (PCI-SIG Industry, 2010) that allows the partitioning of a single Ethernet adapter port into multiple virtual functions. It is apparent that big data systems often have several components, each with distinct hardware requirements. The interaction of the hardware components has an impact on the performance of the applications.

Business Rules and Metadata Stores

The concept of metadata in a data warehouse is not new. Metadata that is complete and allows analysts to understand the context of the raw data will likely lead to higher success rates with data mining projects (Inmon, 1996). Likewise, allowing applications to understand the data quality of data that they are utilizing will allow report creation tools to tag the appropriate reports with confidence metrics based on the data quality. The metadata can also serve to track the progress of data ingestion and data cleaning jobs. For example, data may only be available for use after it has passed through X steps in a data cleaning pipeline. The value of X can also be stored as a business rule, and applications (or a data access layer) can utilize this rule to determine when they can access data. It is conceivable that metadata may need to be shared between organizations and systems. Vetterli et al. (Vetterli, Vaduva, & Staudt, 2000) provides an overview of two industry standards for metadata models, the Open Information Model (OIM) and the Common Warehouse Metamodel (CWM). Metadata can also be utilized to support security policies (Katic, Quirchmay, Schiefer, Stolba, &

Tjoa, 1998) in data warehouses. Metadata of all types forms the basis of the organization's business rules. Ideally, there should be one model to generically represent all types of business rules (Perkins, 2000). Perkins also notes that the business rule process facilitates data sharing because the institutional knowledge is accessible to any application or analyst. Kaula discusses an implementation of a business rule framework in Oracle PL/SQL (Kaula, 2009).

BUSINESS RULES AND METADATA STORES

Businesses are increasingly using data for analysis purposes and revisiting old data sources, such as click-stream data, for integration with other enterprise data sources. The promised benefit of integration is that new insights will emerge from an integrated view of data. Providing more data to decision support algorithms will allow the algorithms to select the most “relevant” attributes from a larger set of attributes for a classification task. Ideally, this will enable the algorithm to build more sophisticated models since data from other data sources may enable discovery of trends that were not discernible in a smaller dataset. For clustering algorithms, click-stream data from multiple websites would potentially enable finer clusters of users to be established. For example, if one website is a foreign policy blog, and another website is a technology blog, correlating data from both sources would ideally allow clustering algorithms to determine finer grained clusters. Perhaps those that read both blogs regularly form one cluster, while another cluster is composed of casual readers of the technology blog, and a third cluster contains casual readers of the foreign policy blog.

Data integration may appear to be a straightforward and obvious task, and indeed it is obvious from a theoretical perspective. Clearly, one major requirement is to ensure that the appropriate data can always be correctly correlated. Revisiting the example above, this integration would require that a common user identity is available across both websites, or at least that it is possible to identify the same unique user in both data sets. Assume for a moment, that the foreign policy blog was created by a political science professor, and the technology blog was created by a technology hobbyist, and both blogs were later acquired by a media company. It is unlikely that both blogs contain a common user identity platform and the media company must attempt to use email addresses, or other proxy attributes for user identity (of course, internet users can have multiple email identities, and therefore this process would likely introduce errors). This idea of correlation may appear to be an obvious and trivial requirement, but in practice, it is a major concern. Based on experiences at several organizations, it is clear that projects are often created and managed within a silo, without regard for enterprise wide integration. Years after the initial systems were designed there is an interest in correlating data across many sources. Each system typically has its own user identity management system, or perhaps even generic group accounts that are utilized by web applications or external organizations. Obviously, discerning user identity mappings between sources within this type of framework may be quite error prone. One approach that could partially alleviate this problem is if each data source provides an interface for other applications or data sources. This interface could be provided by a service or consist of a metadata repository that was easily accessible to other applications.

One trend that was observed across a variety of industries is the need for metadata (or an interface to access other data sources). Many current data warehouse designs resemble the design of databases. Essentially, they contain schemas to describe related objects. The primary assumption, which appears to be false in a dynamic business environment, is that the relationships are static and attribute sets (to represent objects) are static. For example, the assumption that a specific query Q that is executed in year X covering a set of attributes A will also be correct in year Y (where Y is after X), because the business will still have a set of attributes A that map to the same object(s) in the real world. A simple example to illustrate why the same attributes do not often represent objects at later times is below.

Simple Case Study Outlining the Need for Metadata

A university creates a student information system in 1985 and decides to utilize social security numbers to uniquely identify a student. Sometime later, perhaps in the year 2000, when identify theft becomes popular, the university realizes that social security numbers are not ideal identifiers since they may lead to identity theft if the system is compromised or reports are shown to disreputable individuals. They decide to implement a ten digit student identifier, composed of a monotonically increasing integer. Student records from 1985 through 2000 have primary keys of social security numbers. They could generate a student identifier for previous students, but these students may not have any documentation to uniquely identify their records, since they were never assigned a “student identifier” during their time at the university. They may decide to keep the social security number in another table, and allow privileged applications and users to access this data for specific purposes, such as former student records. Various systems within the university may need updated to expect a ten digit identifier, instead of a nine digit social security number for the primary student identification number. The rules to validate a social security number (some area identifiers are not issued and it is impossible to have a social security number containing an unissued area number) are distinct from those that validate a student identifier. In this example, the social security number would be used to lookup records in other tables, such as a student grade table, in the data warehouse that used the social security number as a key from the years 1985-2000. And, the student identifier would be used to lookup records in the student grade table after the year 2000.

Metadata

The benefits of an integrated enterprise business rule (or metadata) repository are significant. The challenge is to determine how to build a general and hierarchical business rule repository that will support a wide variety of applications. The system should be hierarchical (from a logical perspective) because the business will likely have high level rules that will span a variety of lower level rules. For example, if data is missing from a given website for a specific period of time, then the first level rule may be the dataset, followed by a second level rule stating the data quality requirement of a date range, a third level rule noting the date range, and a fourth level rule with a data range. A figure of such a structure is demonstrated in Figure 1.

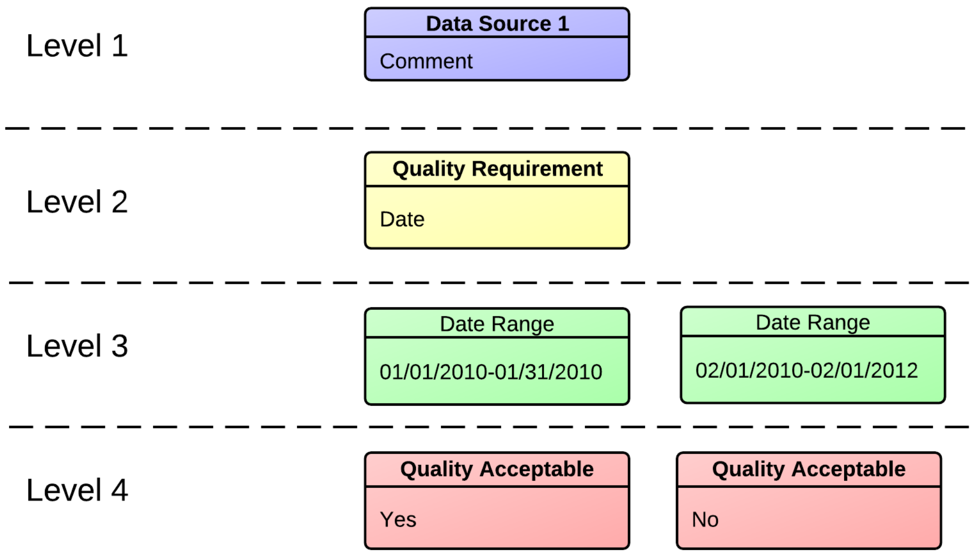

Figure 2. Shows a logical view of a rule that describes data quality.

The advantage of a general and hierarchical business rule framework is that it should be able to support a dynamic business environment without changes to the repository structure. Several rule engines are implemented as in memory systems and can be integrated with other software packages through an API or used as SOA endpoint. Two examples are Drools (The JBoss Drools Team, 2012) and Jess (Friedman-Hill, 2008).

There are several challenges to implementing a business rule repository. For example, how does the need to query the repository along with querying the data warehouse affect the response time for the data warehouse applications? How does the repository designer ensure that the business rule repository is general enough from a structural standpoint? How does the designer preserve the history of rule changes as applied to various sets of data (i.e. apply the 2011 data quality rules to data in 2012 to compare data quality in 2012 to the data quality that was presented in a report in 2011)? Additionally, the mapping of “business rules” to a hierarchical structure has sparse treatment in the literature. The challenge is to build a framework that is general enough to support the dynamic nature of business needs yet also yields appropriate performance. Several rule engines exist that can be incorporated into existing software through API calls or SOA requests.

Metadata Benchmarking Techniques

Determining the appropriate rule engine for a big data project should start with an understanding of the system's architecture. For example, does a lookup in a rule engine need to occur for each row of data that is retrieved? Secondly, do the time constraints permit a SOA procedure call to a remote system? If the time constraints do not permit a lookup, the rule engine may be able to be integrated into the software, data may need to be tagged with rules (of course, tagging data and introducing additional code into the MapReduce job will increase the computation time necessary to process the data and the rules) or the rules engine may be completely separate from the big data system. In the case of the later, the rules may only be consulted as a post processing step to answer ad-hoc questions about the data source.

Big data systems introduce interesting considerations since they are often separate pieces of software that are not extensively integrated with databases and other enterprise middleware such as rule engines. Unstructured and semi-structured data often varies in its quality over time as changes are made to the software generating data, bugs are discovered and corrected, or the data loading and transformation workflow breaks down. Structured data can also change over time as attributes are added or removed, the meaning of an attribute may change (for example, a customer may change the unit of measurement from inches to centimeters), or breakdowns in the data loading and transformation system may occur.

For example, calculating the “average length” of a widget requires that the numbers contained in an attribute containing length in a data system have a consistent unit across the records that are being used to calculate “average length.” As part of the data loading process, data could be converted to a standard unit of length at the time of data load or converted during computation if a rule engine could be consulted and the conversion, if any, could be applied. In this simple case, the organization may decide to correct the data at the time of data loading. However, situations arise when it is necessary to retain original data from customers in a data warehouse and transform the raw data into clean data in a data mart or when data is lost. In situations such as these, it may be necessary to know that data is missing or not available for computation so that correct results are computed. For example, if 10% of a click behavior data feed is missing for a given day X then simply counting these records in the data mart would not allow a comparison of the actual traffic for days X & Y.

The benchmarking of a Rete algorithm based rule engine, such as Jess or Drools, should be performed with actual rules and data since the runtime of the algorithm depends on the specific rules and data used (Friedman-Hill, 2008). The performance of a rule engine based on database records and stored procedures would depend on the layout of the tables, attributes, indexing structure and the algorithm for the rule checking. In memory databases may prove useful if the goal is provide a lower response time than a database based on traditional magnetic disks or solid state disks.

Given the apparent lack of accepted big data benchmarking methodologies, rules engines add to the complexity of constructing a benchmark for modern big data systems. If possible, representative tasks and data should be used to determine if the rules engine or metadata repository can satisfy the requirements.

The use of metadata in data warehouses appears to be a novel idea in practice (theoretically, it has been recognized for some time). Additional research needs to be carried out to categorize the types of workloads that modern data warehouses service so that appropriate benchmarks can be built that consider metadata. Regrettably, since data warehouses contain organizations' sensitive data, organizations are often reluctant to release any information about the data warehouse or its architecture. Perhaps a more important problem is that design standards for data warehouse metadata don't appear to be adopted in practice.

BENCHMARKING

Big data systems are unique for several reasons. First, they typically transfer a large volume of data in large chunks and do so for an extended period of time (i.e. a large MapReduce job processing several TB of data). Secondly, production big data systems, such as a MapReduce cluster, are typically purpose built and not shared with general purpose computing workloads. I/O performance becomes critically important when building big data systems. To maximize the I/O throughput, the motherboard and chipset should support adequate dedicated PCI-Express bandwidth for hardware peripherals such as network and storage devices. The chipset should also support adequate bandwidth between the PCI-Express bus and the CPU and memory. Some motherboard manufacturers make this information available in their manuals. When performing benchmarks of storage or network devices, the benchmark parameters should be selected carefully. Benchmarks of network devices should include a range of packet sizes around the packet size that is typically used in the target application workload. This will ensure that the desired throughput and latency is achievable at the packet size that is used in the target workload. Benchmarks of storage systems should include read and write patterns that closely mirror that workload of the target application. The goal of these benchmarks is to establish an upper bound on the performance of the hardware under ideal conditions. As an added benefit, benchmarks of hardware devices may expose stability issues with drivers or incompatibilities with the motherboard or other hardware components.

After this phase is complete, an application level benchmark should be completed to ensure that the operating system, CPU, and memory can effectively achieve the desired performance while performing extensive I/O activity. Various benchmarks are available for specific big data applications such as MapReduce. If virtualized compute nodes are utilized, it would be appropriate to perform hardware micro-benchmarks and application benchmarks at the native operating system level and then repeat this process in the virtualized environment. The benchmarks at the native operating system level establish an upper bound on performance. The benchmarks in the virtualized environment will allow the designer to determine the level of overhead that is present as a result of the virtualized environment.

Storage systems often need to be highly stable under prolonged periods of load in big data systems. Therefore, the selection of a stable yet high performance storage controller is an important consideration if the system will rely on local storage. Additionally, network controllers are likely to be utilized extensively for applications such as a Hadoop file system. If the storage for the big data application is accessed over a network, it is advisable to ensure that if only one network controller is used, it can support the bandwidth necessary to read data from a network file system, and perform the network communication that is necessary for the big data application. Some designers may decide to create a network for network based storage and another network for cluster communication and devote a team of network ports or even a controller to each network. If the big data application is running in a virtualized environment, PCI-Passthrough (Jones, 2009) and PCI-SIG SR-IOV (PCI-SIG Industry, 2010) enables the allocation of devices or virtualized devices (at the hardware level), respectively, to virtual machines.

Benchmarking Case Study

A car manufacture wants to predict the failure of products in the field using data from component vendors, repair shops, and internal manufacturing databases. The internal databases allow the manufacturer to determine which components were used to create each product. The data warehouse will need to continuously load data from these sources. The benchmarking strategy should begin with an estimation of the volume of data that needs loaded in a given time frame and the estimated storage volume of the data warehouse. Then, the system designer should consider whether a cloud environment (private or public) or a traditional operating system installed in a local data center is appropriate. Considerations such as cost, fault-tolerance, and network bandwidth and network reliability will influence the decision of whether to utilize an existing local data center or utilize a public or private cloud.

Cloud implementations utilize virtualization to achieve cost efficiency (by placing numerous virtual machines on one physical server or enabling virtual machines to be quickly provisioned on a physical server) but the I/O efficiency of virtual environments is lower than an application executing on an operating system executing directly on the hardware. If a cloud implementation is utilized, then the I/O virtualization overheard should be evaluated, especially for storage and networking devices, and the data warehouse product should be evaluated to ensure that it is certified by the vendor for use in a virtualized environment. Then, testing of the data warehouse application can be conducted. These tests should determine if the data warehouse can support the desired level of data loading performance and query performance (based on estimated use). If multiple nodes are required to achieve the target level of performance, then multiple nodes should be utilized in the test. If multiple nodes are utilized in a virtual environment, networking benchmarks between virtual nodes will give an approximation of the best-case scalability that can be achieved across multiple nodes. Basic fault-tolerance of the application can be tested by disconnecting the node from the network, halting the node, or removing disks from the storage system to ensure that the application can still function under a variety of hardware failure conditions. More detailed or advanced fault-tolerance testing could be carried out, but fault-tolerance testing in a cloud computing environment is not significantly theoretically distinct from fault-tolerance testing of other types of distributed systems. For example, Tannenbaum and Van Steen's book devotes a chapter to a comprehensive discussion of fault-tolerance in distributed systems (Tannenbaum & Van Steen, 2007).

FUTURE RESEARCH DIRECTIONS

The benchmarking of big data systems appears to be primarily focused on specific classes of systems, such as MapReduce clusters, databases, or distributed storage. While it is certainly important to understand the baseline performance of each component in a system, it is also important to understand how a system will behave in production when it is built from various components. An important opportunity for increased focus may be the classification of big data workloads to determine if there are common types of workloads. For example, are most batch analysis workloads similar irrespective of whether MapReduce or another tool is used? Do most reporting applications place similar demands on the data sources and metadata stores? What is the typical response time expected from a batch analysis system? What type of demands do big data applications, such as ecommerce sites, place on components? Answering these types of questions would allow a characterization of a “big data” workload in a modern organization and allow tool builders to determine which specific systems produce the optimal mix of performance and cost for a given workload. Is it possible to use one analysis tool, such as MapReduce, to support the workload, or must the big data system be composed of various components? While basic concepts suggests that algorithm X should be implemented in system Y, or system A can only scale to B terabytes, design of big data systems in the real world appears to be guided by heuristics rather than more formal scientific processes.

CONCLUSION

Big data systems can include data collection systems, data cleaning, data processing, data analysis, and applications for reporting. Big data systems typically process large amounts of data in a distributed fashion and may rely on accessing metadata to produce accurate results. Big data systems are composed of many components and each component often requires multiple nodes and extensive storage. Therefore, these types of systems are relatively costly to design and maintain, whether they are capital purchases or cloud computing based. The choice of utilizing applications in a native operating system environment or in a cloud computing environment will likely depend on the infrastructure capabilities of the organization, the respective costs, and the requirements of the project. Cloud computing is typically implemented with virtual machine technologies and the choice of a hypervisor has an impact on the performance of the system, especially in regard to I/O. Since big data systems often rely on I/O capacity, the performance of I/O subsystems is important. The chapter presents several issues that emerge when attempting to use metadata stores and cloud computing to support “big data” projects.

References

Amazon Web Services, Inc. (n.d.). *Amazon EC2 Instance Types*. Retrieved January 19, 2013, from http://aws.amazon.com/ec2/instance-types/

Bhatia, N. (2009). *Performance Evaluation of AMD RVI Hardware Assist.* VMWare, Inc.

Friedman-Hill, E. J. (2008, November 5). *Jess, the Rule Engine for the Java Platform - The Rete Algorithm*. Retrieved January 26, 2013, from http://www.jessrules.com/jess/docs/71/rete.html

Inmon, W. H. (1996). The data warehouse and data mining. *Commun. ACM* , 49--50.

Jones, T. (2009, October). *Linux virtualization and PCI passthrough*. Retrieved May 2012, from http://www.ibm.com/developerworks/linux/library/l-pci-passthrough/

Katic, N., Quirchmay, G., Schiefer, J., Stolba, M., & Tjoa, A. (1998). A prototype model for data warehouse security based on metadata. *Ninth International Workshop on Database and Expert Systems Applications, 1998.*, (pp. 300-308).

Kaula, R. (2009). Business Rules for Data Warehouse. *International Journal of Information and Communication Engineering* , 359-367.

Nagarajan, A. B., Mueller, F., Engelmann, C., & Scott, S. L. (2007). Proactive Fault Tolerance for HPC with Xen Virtualization. *Proceedings of the 21st annual international conference on Supercomputing* (pp. 23-32). Seattle: ACM.

PCI-SIG Industry. (2010, January). *Single Root I/O Virtualization and Sharing 1.1 Specification*. Retrieved November 2010, from http://www.pcisig.com/specifications/iov/single_root/

Perkins, A. (2000). Business rules=meta-data. *34th International Conference on Technology of Object-Oriented Languages and Systems, 2000*, (pp. 285-294).

Regola, N., & Ducom, J.-C. (2010). Recommendations for Virtualization Technologies in High Performance Computing. *IEEE International Conference on Cloud Computing Technology and Science* (pp. 409-416). Indianapolis: IEEE.

Tannenbaum, A., & Van Steen, M. (2007). Fault Tolerance. In A. Tannenbaum, & M. Van Steen, *Distributed Systems: Principles and Paradigms* (pp. 321-375). Upper Saddle River, NJ: Pearson Education.

The JBoss Drools Team. (2012, November 13). *Drools Introduction and General User Guide*. Retrieved January 26, 2013

Vetterli, T., Vaduva, A., & Staudt, M. (2000). Metadata standards for data warehousing: open information model vs. common warehouse metadata. *SIGMOD Rec.* , 68-75.

ADDITIONAL READING

A. Menon, A., Santos, J., Turner, Y., Janakiraman, G., & Zwaenepoel, W. (2005). Diagnosing performance overheads in the Xen virtual machine environment. *International Conference on Virtual Execution Environment.*

Armbrust, M., Fox, A., o, Griffith, R., Joseph, A. D., Katz, R., et al. (2010). A view of cloud computing. *Commun. ACM* , 50--58.

Bailey, D., Harris, T., Saphir, W., Van Der Wijngaart, R., Woo, A., & Yarrow, M. (1995). *NAS parallel benchmarks 2.0.* NASA Ames Research Center.

Bennett, C., Grossman, R. L., Locke, D., Seidman, J., & Vejcik, S. (2010). Malstone: towards a benchmark for analytics on large data clouds. *16th ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 145-152). New York: ACM.

Chen, Y. (2012). We Don't Know Enough to make a Big Data Benchmark Suite An Academia-Industry View. *Second Workshop on Big Data Benchmarking.* Pune, India.

Chu, C.-T., Kim, S. K., Lin, Y.-A., Yu, Y., Bradski, G., Ng, A. Y., et al. (2007). Map-Reduce for Machine Learning on Multicore. In B. Scholkopf, J. Platt, & T. Hoffman, *Advances in Neural Information Processing Systems 19* (pp. 281-288). MIT Press.

Dean, J., & Ghemawat, S. (2004). Mapreduce: simplified data processing on large clusters. *OSDI.04: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation.* USENIX Association.

Figueiredo, R., Dinda, P. A., & Fortes, J. (2003). A case for grid computing on virtual machines. *3rd International Conference on Distributed Computing Systems* (p. 550). Washington: IEEE Computer Society.

Grossman, R., & Gul, Y. (2008). Data mining using high performance data clouds: experimental studies using sector and sphere. *14th ACM SIGKDD international conference on Knowledge discovery and data mining, KDD '08* (pp. 920-927). New York, NY: ACM.

Jian, Z., Xiaoyong, L., & Haibing, G. (2008). The optimization of Xen network virtualization. *CSSE '08: Proceedings of the 2008 International Conference on Computer Science and Software Engineering* (pp. 431-436). Washington, DC: IEEE Computer Society.

Juve, G., Deelman, E., Vahi, K., Mehta, G., Berriman, B., Berman, B., et al. (2009). Scientific workflow applications on Amazon EC2. *2009 5th IEEE International Conference on E-Science Workshops* (pp. 59-66). IEEE.

Kontagora, M., & Gonzalez-Velez, H. (2010). Benchmarking a mapreduce environment on a full virtualisation platform. *Proceedings of the 2010 International Conference on Complex, Intelligent and Software Intensive Systems* (pp. 433-438). Washington, DC: IEEE Computer Society.

Lee, S.-W., Moon, B., & Park, C. (2009). Advances in Flash Memory SSD Technology for Enterprise Database Applications. *35th SIG-MOD international conference on Management of data* (pp. 863-870). New York: ACM.

Lee, S.-W., Moon, B., Park, C., Kim, J.-M., & Kim, S.-W. (2008). A case for flash memory ssd in enterprise database applications. *2008 ACM SIGMOD international conference on Management of data* (pp. 1075-1086). New York: ACM.

Liu, J., Huang, W., Abali, B., & Panda, D. K. (2006). High performance VMM-bypass I/O in virtual machines. *TEC '06: Proceedings of the annual conference on USENIX '06 Annual Technical Conference* (p. 3). Berkeley, CA: USENIX Association.

Menon, A., Cox, A. L., & Zwaenepoel, W. (2006). Optimizing network virtualization in xen. *USENIX Annual Technical Conference*, (pp. 15-28).
Nagle, D., Serenyi, D., & Matthews, A. (2004). The Panasas ActiveScale Storage Cluster: Delivering Scalable High Bandwidth Storage. *2004 ACM/IEEE conference on Supercomputing* (p. 53). Washington: IEEE Computer Society.
Nussbaum, L., Anhalt, F., Mornard, O., & Gelas, J.-P. (2009). Linux-based virtualization for HPC clusters. *Linux Symposium*, (pp. 221-234).
Padala, P., Zhu, X., Wang, Z., Singhal, S., & Shin, K. G. (2007). *Performance evaluation of virtualization technologies for server consolidation.* HP Labs.
Pavlo, A., Paulson, E., Rasin, A., Abadi, D. J., DeWitt, D. J., Madden, S., et al. (2009). A comparison of approaches to large-scale data analysis. *35th SIGMOD international conference on Management of data, SIGMOD '09* (pp. 165-178). New York: ACM.
Ranger, C., Raghuraman, R., Penmetsa, A., Bradski, G., & Kozyrakis, C. (2007). Evaluating mapreduce for multi-core and multiprocessor systems. *PCA .07: Proceedings of the 13th International Symposium on High-Performance Computer Architecture* (pp. 13-24). IEEE Computer Society.
Wang, G., & Ng, T. (2010). The Impact of Virtualization on Network Performance of Amazon EC2 Data Center. *2010 IEEE INFOCOM* (pp. 1-9). IEEE.
Wu, J., Ping, L., Ge, X., Wang, Y., & Fu, J. (2010). Cloud Storage as the Infrastructure of Cloud Computing. *2010 International Conference on Intelligent Computing and Cognitive Informatics (ICICCI)* (pp. 380-383). IEEE.
Youseff, L., Wolski, R., Gorda, B., & Krintz, C. (2006). Evaluating the performance impact of Xen on MPI and process execution in HPC systems. *Virtualization Technology in Distributed Computing.* Washington: IEEE Computer Society .
Zaharia, M., Konwinski, A., Joseph, A. D., Katz, R., & Stoica, I. (2008). Improving mapreduce performance in heterogeneous environments. *8th USENIX conference on Operating systems design and implementation, OSDI'08* (pp. 29-42). Berkeley, CA: USENIX Association.

KEY TERMS AND DEFINITIONS

Benchmarking: an experimental method to determine the performance of a system for a range of operations (e.g. disk read, disk write, disk random read, and disk random write) or an application workflow.

Cloud computing: a generic term utilized to describe leased computer resources including storage, networking, backup, application runtimes, and clusters.

Data warehouse: a generic description of an enterprise data store that is utilized to store data that is utilized to operate a business.

MapReduce: a paradigm of data processing or the specific paper that introduced the implementation.

Microbenchmark: an experimental method to determine the performance of a specific type of hardware device or a specific operation (e.g. disk read)

Overhead: loss in efficiency as a result of an intermediate stage that is not present in the “maximally efficient” design

Virtual machine: a generic name to denote support for running guest operating systems, irrespective of the type of implementation.