THE HADOOP STACK: NEW PARADIGM FOR BIG DATA STORAGE AND PROCESSING

Contributors

Jinquan Dai
Software and Services Group,
Intel Corporation

Jie Huang
Software and Services Group,
Intel Corporation

Shengsheng Huang
Software and Services Group,
Intel Corporation

Yan Liu
Software and Services Group,
Intel Corporation

Yuanhao Sun
Software and Services Group,
Intel Corporation

We are on the verge of the "industrial revolution of Big Data," which represents the next frontier for innovation, competition, and productivity.[1]. Big data is rich with promise, but equally rife with challenges—it extends beyond traditional structured (or relational) data, including unstructured data of all types; it is not only large in size, but also growing faster than Moore's law. In this article, we first present the new paradigm (in particular, the Hadoop stack) that is required for big data storage and processing. After that, we describe how to optimize the Hadoop deployment through proven methodologies and tools provided by Intel (such as HiBench and HiTune). Finally, we demonstrate the challenges and possible solutions for real-world big data applications using a case study of an intelligent transportation system (ITS) application.

Introduction

We are on the verge of the "industrial revolution of Big Data," where a vast range of data sources (from web logs and click streams, to phone records and medical history, to sensors and surveillance cameras) is flooding the world with enormous volumes of data in a huge variety of different forms. Significant values are being extracted from this data deluge with extremely high velocity. Big data is already the heartbeat of Internet, social and mobile; more importantly, it is heading toward ubiquity as enterprises (telecommunications, governments, financial services, healthcare, and so on) have amassed terabytes and even petabytes of data that they are not yet prepared to process. And soon, all will be awash with even more data from ubiquitous devices and sensors as we enter the age of the Internet of Things. These big data trends represent the next frontier for innovation, competition, and productivity.

Big data is rich with promise, but equally rife with challenges. It extends beyond traditional structured (or relational) data, including unstructured data of all types (text, image, video, and more); it is not only large in size, but also growing faster than Moore's law (more than doubling every two years).[2] In this article, we first present the new paradigm (in particular, the Hadoop stack) that is required for big data storage and processing. After that, we describe how to optimize the Hadoop deployment (through proven methodologies and tools provided by Intel). Finally, we demonstrate the challenges and possible solutions for real-world big data applications using a case study.

New Paradigm for Big Data Analytics

Big data is powering the next industrial revolution. Pioneering Web companies (such as Google, Facebook, Amazon, and Taobao) have already been looking

at data in completely new ways to improve their business. It will become even more critical for corporations and governments to harvest values from the massive amount of untapped data in the future.

However, big data is also radically different from traditional data. In this section, we describe the new challenges brought by big data and the new paradigm for big data processing that addresses these challenges.

Big Data Is Different from Traditional Data

Some define big data as the "datasets whose size is beyond the ability of typical database software tools to capture, store, manage and analyze."[1] However, big data is not only massive in scale, but also diverse in nature.

- *Unstructured data:* Unlike traditional structured (or relational) data in enterprise databases or data warehouses, big data is mostly about unstructured data—coming from many different sources, in many different forms (such as text, image, audio, video, and sensor readings), and often with conflicting syntax and semantics.
- *Massive scale and growth:* Unstructured data is growing 10–50 times faster than structured data, and soon will represent 90 percent of all data.[2] Consequently, big data is both large in size (10–100 times larger than traditional data warehouses[3]) and growing exponentially (increasing by roughly 60 percent annually), faster than Moore's law.[2]
- *Scale-out framework:* The massive scale, exponential growth, and viable nature of big data necessitate a much more scalable and flexible data management and analytics framework. Consequently, emerging big data frameworks (such as Hadoop/MapReduce and NoSQL) have adopted a scale-out (rather than scale-up), shared-nothing architecture, with massively distributed software running on clusters of independent servers.
- *Real-time, predictive analytics:* There is significant value to be extracted from big data (for example, a 300 billion US dollar [USD] potential annual value to US healthcare[1]). To realize this value, a new class of predictive analytics (with complex machine learning, statistic modeling, graph analysis, and so on) is needed to identify the future trends and patterns from within massive seas of data and in (near) real time as data is streaming in continuously.

The Hadoop Stack: A New Big Data Processing Paradigm

The massive scale, exponential growth, and variable nature of big data necessitate a new data processing paradigm. In particular, the Hadoop stack, as illustrated in Figure 1, has emerged as the de facto standard for big data storage and processing. In this section, we provide a brief overview of the core components in the Hadoop stack (namely, MapReduce[4][5], HDFS[5][6], Hive[7][8], Pig[9][10], and HBase[11][12]).

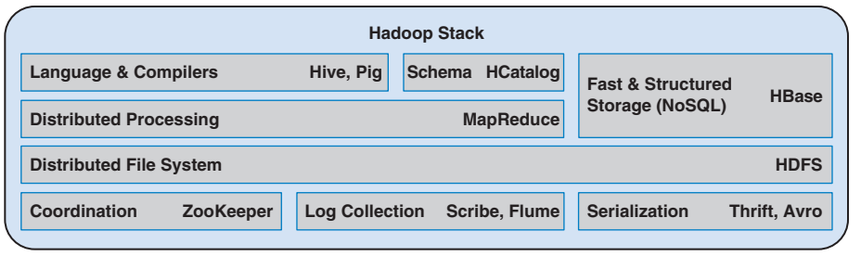

Figure 1: The Hadoop stack
(Source: Intel Corporation, 2012)

MapReduce: Distributed Data Processing Framework

At a high level, the MapReduce[4] model dictates a two-stage group-by-aggregation dataflow graph to the users, as shown in Figure 2. In the first phase, a *Map* function is applied in parallel to each partition of the input data, performing the grouping operations. In the second phase, a *Reduce* function is applied in parallel to each group produced by the first phase, performing the final aggregation. In addition, a user-defined *Combiner* function can be optionally applied to perform the map-side "pre-aggregation" in the first phase, which helps decrease the amount of data transferred between the map and reduce phases.

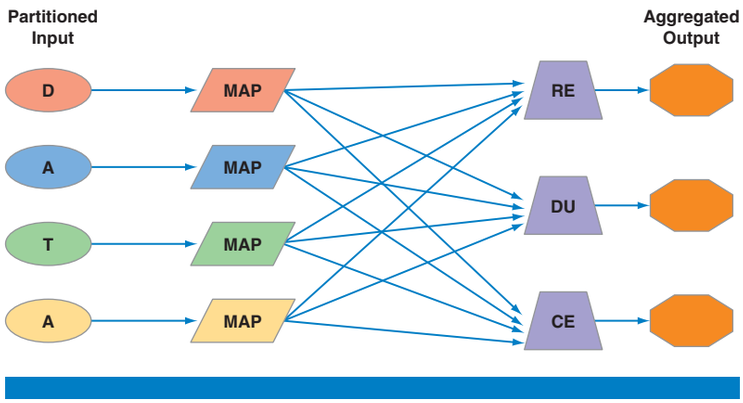

Figure 2: MapReduce dataflow model
(Source: Intel Corporation, 2012)

Hadoop is a popular open source implementation of MapReduce. The Hadoop framework is responsible for running a MapReduce program on the underlying cluster that may comprise thousands of nodes. In the Hadoop cluster, there is a single master (*JobTracker*) controlling a number of slaves (*TaskTrackers*). The user writes a Hadoop job to specify the Map and Reduce functions (in addition to other information such as the locations of the input and output), and submits the job to the JobTracker. The Hadoop framework divides a Hadoop job into a series of map or reduce tasks; the master is responsible for assigning

the tasks to run concurrently on the slaves, and rescheduling the tasks upon failures. Figure 3 illustrates the architecture of the Hadoop cluster.

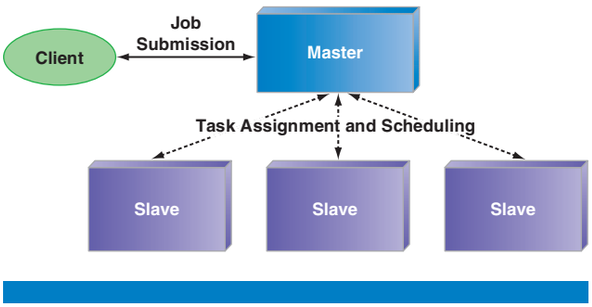

Figure 3: Architecture of the Hadoop cluster
(Source: Intel Corporation, 2012)

Hive and Pig: High-Level language for Distributed Data Processing

The Pig[9] and Hive[7] systems allow the users to perform ad-hoc analysis of big data on top of Hadoop, using dataflow-style scripts and SQL-like queries respectively. For instance, The following example shows the Pig program (an example in the original Pig paper[9]) and Hive query for the same operation (that is, finding, for each sufficiently large category, the average *pagerank* of high-pagerank *urls* in that category). In these two systems, the high level query or script program is automatically compiled into a series of MapReduce jobs that are executed on the underlying Hadoop system.

```
Pig Script
good_urls = FILTER urls BY pagerank > 0.2;
groups = GROUP good_urls BY category;
big_groups = FILTER groups BY COUNT(good_urls)>1000000;
output = FOREACH big_groups GENERATE category, AVG(good_urls.pagerank);
```

```
Hive Query
SELECT category, AVG(pagerank)
FROM (SELECT category, pagerank, count(1) AS recordnum
             FROM urls WHERE pagerank > 0.2
             GROUP BY category) big_groups
WHERE big_groups.recordnum > 1000000
```

AU: We have renumbered figure No. from here onwards. Please check.

HDFS: Hadoop Distributed File System

The Hadoop distributed file system (HDFS) is a popular open source implementation of Google File System.[6] An HDFS cluster consists of a single master (or *NameNode*) and multiple slaves (or *DataNodes*), and is accessed by multiple clients, as illustrated in Figure 4. Each file in HDFS is divided into large (64 MB by default) blocks and each block is replicated on multiple (3 by default) DataNodes.

The NameNode maintains all file system metadata (such as the namespace and replica locations), and the DataNodes store HDFS blocks in local file systems and

handle HDFS read/write requests. An HDFS client interacts with the NameNode for metadata operations (for example, open file or delete file) and replica locations, and it directly interacts with the appropriate DataNode for file read/write.

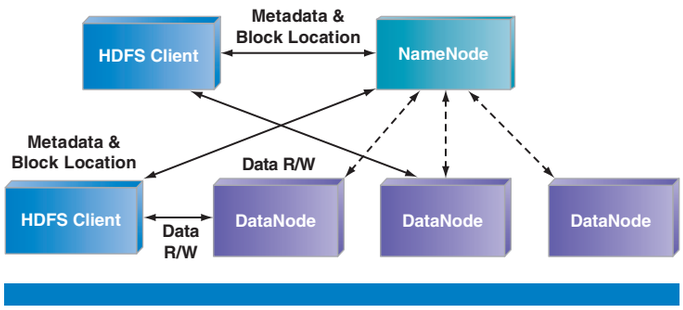

Figure 4: HDFS Architecture
(Source: Intel Corporation, 2012)

HBase: High Performance, Semi-Structured (NOSQL) Database

HBase, an open source implementation of Google's Bigtable[11], provides a sparse, distributed, column-oriented table store. In HBase, each value is indexed by the tuple (row, column, timestamp), as illustrated in Figure 5. The HBase API provides functions for looking up, writing, and deleting values using the specific row key (possibly with column names), and for iterating over data in several consecutive rows.

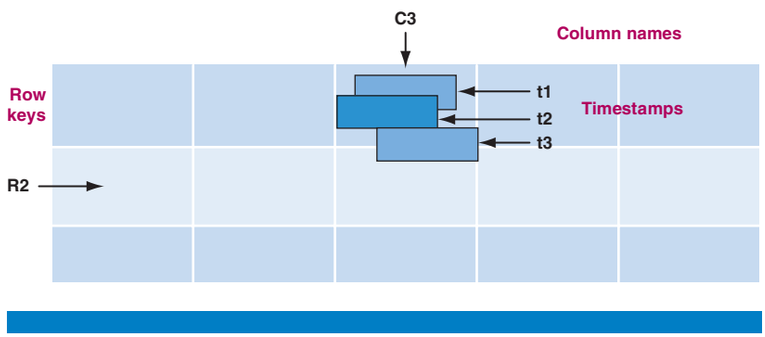

Figure 5: The HBase Data Model
(Source: Intel Corporation, 2012)

Physically, rows are ordered lexicographically and dynamically partitioned into row ranges (or *regions*). Each region is assigned to a single *RegionServer*, which handles the all data accesses requests to its regions. Mutations are first committed to the append-only log on HDFS and then write to the in-memory *memtable* buffer; the data in the region are stored in SSTable[11] files (or *HFiles*) on HDFS, and a read operation is executed on a merged view of the sequence of HFiles and the memtable.

Optimizing Hadoop Deployments

As Hadoop-based big data systems grow in pervasiveness and scale, efficiently tuning and optimizations of Hadoop deployments remain a huge challenge for the users. For instance, within the Hadoop community, tuning Hadoop jobs is considered to be a very difficult problem and requires a lot of effort to understand Hadoop internals[13]; in addition, the lack of tuning tools for Hadoop often forces users to resort to trial-and-error tuning.[14] In this section, we present systematic approaches to optimizing Hadoop deployments, including representative workloads, dataflow-based analysis, and a performance analyzer for Hadoop.

HiBench: A Representative Hadoop Benchmark Suite

To understand the characteristics of typical Hadoop workloads, we have constructed HiBench[15], a representative and comprehensive benchmark suite for Hadoop, which consists of a set of Hadoop programs including both synthetic micro-benchmarks and real-world applications. Currently the HiBench suite contains ten workloads, classified into four categories, as shown in Table 1.

Category	Workload
Micro Benchmarks	Sort
	WordCount
	TeraSort
	EnhancedDFSIO
Web Search	Nutch Indexing
	Page Rank
Machine Learning	Bayesian Classification
	K-means Clustering
Analytical Query	Hive Join
	Hive Aggregation

Table 1: HiBench Workloads
(Source: Intel Corporation, 2012)

Micro Benchmarks

The *Sort*, *WordCount,* and *TeraSort*[16] programs contained in the Hadoop distribution are three popular micro-benchmarks widely used in the community, and therefore are included in HiBench. Both the Sort and WordCount programs are representative of a large subset of real-world MapReduce jobs—one transforming data from one representation to another, and another extracting a small amount of interesting data from a large data set.

In HiBench, the input data of Sort and WordCount workloads are generated using the RandomTextWriter program contained in the Hadoop distribution. The TeraSort workload sorts 10 billion 100-byte records generated by the *TeraGen* program contained in the Hadoop distribution.

We have also extended the *DFSIO* program contained in the Hadoop distribution to evaluate the aggregated bandwidth delivered by HDFS. The

original DFSIO program only computes the average I/O rate and throughput of each map task, and it is not a straightforward process to properly sum up the I/O rate or throughput if some map tasks are delayed, retried or speculatively executed by the Hadoop framework. The Enhanced DFSIO workload included in HiBench computes the aggregated bandwidth by sampling the number of bytes read/written at fixed time intervals in each map task; during the reduce and post-processing stage, the samples of each map task are linear interpolated and resampled at a fixed plot rate, so as to compute the aggregated read/write throughput by all the map tasks.[15]

Web Search

The *Nutch Indexing* and *Page Rank* workloads are included in HiBench, because they are representative of one of the most significant uses of MapReduce (that is, large-scale search indexing systems).

The Nutch Indexing workload is the indexing subsystem of Nutch[17], a popular open-source (Apache) search engine; we have used the crawler subsystem in Nutch to crawl an in-house Wikipedia mirror and generated about 2.4 million Web pages as the input of this workload. The Page Rank workload is an open source implementation of the page-rank algorithm in Mahout[18] (an open-source machine learning library built on top of Hadoop).

Machine Learning

The *Bayesian Classification* and *K-means Clustering* implementations contained in Mahout are included in HiBench, because they are representative of one of another important uses of MapReduce (that is, large-scale machine learning).

The Bayesian Classification workload implements the trainer part of Naive Bayesian (a popular classification algorithm for knowledge discovery and data mining). The input of this benchmark is extracted from a subset of the Wikipedia dump. The Wikipedia dump file is first split using the built-in *WikipediaXmlSplitter* in Mahout, and then prepared into text samples using the built-in *WikipediaDatasetCreator* in Mahout. The text samples are finally distributed into several files as the input of the benchmark.

The K-means Clustering workload implements K-means (a well-known clustering algorithm for knowledge discovery and data mining). Its input is a set of samples, and each sample is represented as a numerical d-dimensional vector. We have developed a random data generator using statistic distributions to generate the workload input.

Analytic Query

The *Hive Join* and *Hive Aggregation* queries in the *Hive performance benchmarks*[19] are included in HiBench, because they are representative of another one of the most significant uses of MapReduce (that is, OLAP-style analytical queries).

Both Hive Join and Aggregation queries are adapted from the query examples in Pavlo et. al.[14] They are intended to model complex analytic queries over

structured (relational) tables—Hive Aggregation computes the sum of each group over a single read-only table, while Hive Join computes the both the average and sum for each group of records by joining two different tables.

Data Compression

Data compression is aggressively used in real-world Hadoop deployments, so as to minimize the space used to storing the data, and to reduce the disk and network I/O in running MapReduce jobs. Therefore, each workload in HiBench (except for Enhance DFSIO) can be configured to run with compression turned on or off. When compression is enabled, both the input and output of the workload will be compressed (employing a user-specified codec), so as to evaluate the Hadoop performance with intensive data compressions.

Characterization of Hadoop Workloads Using a Dataflow Approach

In this section we present the Hadoop dataflow model, which provides a powerful framework for the characterizations of Hadoop workloads. It allows the users to understand the application runtime behaviors (such as task scheduling, resource utilizations, and system bottlenecks), and effectively conduct performance analysis and tuning for the Hadoop framework.

Hadoop Dataflow Model

The Hadoop framework is responsible for mapping the abstract MapReduce model (as described earlier) to the underlying cluster, running the input MapReduce program on thousands of nodes in a distributed fashion. Consequently, the Hadoop cluster often appears as a big black box to the users, abstracting away the messy details of data partitioning, task distribution, fault tolerance, and so on. Unfortunately, this abstraction makes it very difficult, if not impossible, for the users to understand the runtime behavior of Hadoop applications.

Based on the Hadoop framework, we have developed a dataflow model (as shown in Figure 6) for the characterization of Hadoop applications. In the Hadoop framework, the input data are first partitioned into *splits*, and then a distinct map task is launched to process each split. Inside each map task, the *map* stage applies the Map function to the input; the *spill* stage divides the intermediate output into several partitions, combines the intermediate output (if a Combiner function is specified), and finally stores the output into temporary files on the local file system.

A distinct reduce task is launched to process each partition of the map outputs. The reduce task consists of three stages (namely, the *shuffle*, *sort,* and *reduce* stages). In the shuffle stage, several parallel copy threads (or *copiers*) fetch the relevant partition of the outputs of all map tasks via HTTP (there is an HTTP server running on each node in the Hadoop cluster); in the meantime, two *merge* threads in the shuffle stage merge those map outputs into temporary files on the local file system. After the shuffle stage is done, the *sort* stage merges all the temporary files, and finally the *reduce* stage processes the partition (that is, applying the Reduce function) and generates the final results.

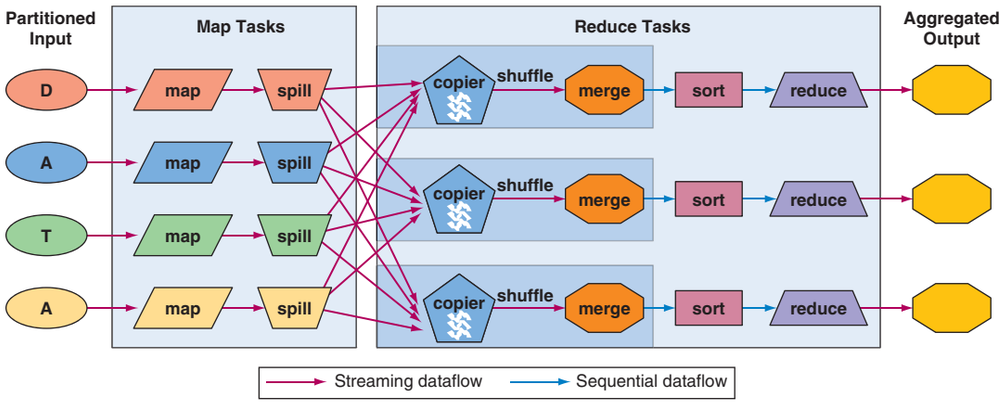

Figure 6: The Hadoop dataflow model
(Source: Intel Corporation, 2012)

Performance Analysis

Figure 7 shows the characteristics of Hadoop TeraSort using the dataflow-based approach. In these charts, the x-axis represents the time elapsed. The first chart in Figure 7 illustrates the timeline-based dataflow execution chart for the workload, where the "bootstrap" line represents the period before the map or reduce task is launched, the "idle" line represents the period after the map or reduce task is complete, the "map" line represents the period when the map tasks are running, and the "shuffle," "sort," and "reduce" lines represent the periods when the corresponding stages are running. The other charts in the figure show the timeline-based CPU, disk, and network utilizations (as sampled by the sysstat package every second) of the slaves respectively. We stack these charts together in one figure, so as to understand the system behaviors during different stages of the Hadoop applications.

As described earlier, TeraSort is a standard benchmark that that sorts 10 billion 100-byte records. It needs to transform a huge amount of data from one representation to another, and therefore is I/O bound in nature. In order to minimize the disk and network I/O during shuffle, we have compressed the map outputs in the experiment, which greatly reduces the shuffle size. Consequently, TeraSort has very high CPU utilization and moderate disk I/O during the map tasks, and moderate CPU utilization and heavy disk I/O during the reduce stages, as shown in Figure 1. In addition, it has almost zero network utilization during the reduce stages, because the final results are not replicated (as required by the benchmark).

HiTune: Hadoop Performance Analyzer

We have built HiHune[20], a dataflow-based Hadoop performance analyzer that allows users to understand the runtime behaviors of Hadoop applications for performance analysis.

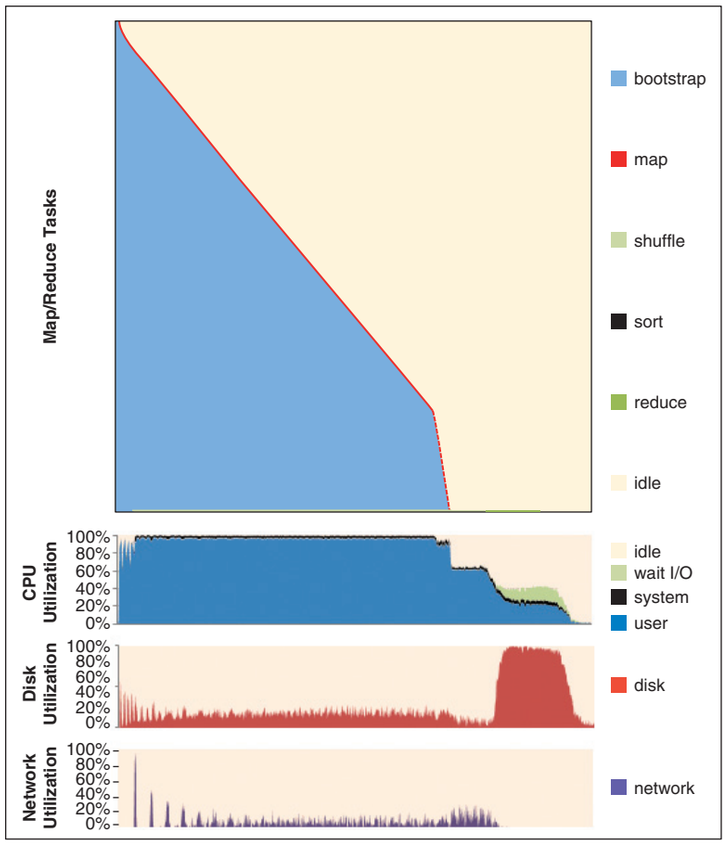

Figure 7: Dataflow-based performance analysis of TeraSort (Source: Intel Corporation, 2012)

Challenges in Hadoop Performance Analysis

As described before, the Hadoop dataflow abstraction makes it very difficult, if not impossible, for users to efficiently provision and tune these massively distributed systems. Performance analysis for the Hadoop application is particularly challenging due to its unique properties.

- *Massively distributed systems:* Each Hadoop application is a complex distributed application, which may comprise tens of thousands of processes and threads running on thousands of machines. Understanding system behaviors in this context would require correlating concurrent performance activities (such as CPU cycles, retired instructions, and lock contentions) with each other across many programs and machines.
- *High level abstractions:* Hadoop allows users to work at an appropriately high level of abstraction, by hiding the messy details of parallelisms behind the dataflow model and dynamically instantiating the dataflow graph (including resource allocations, task scheduling, fault tolerance, and so on). Consequently, it is very difficult, if not impossible, for users to understand how the low level performance activities can be related to the high level abstraction (which they have used to develop and run their applications).

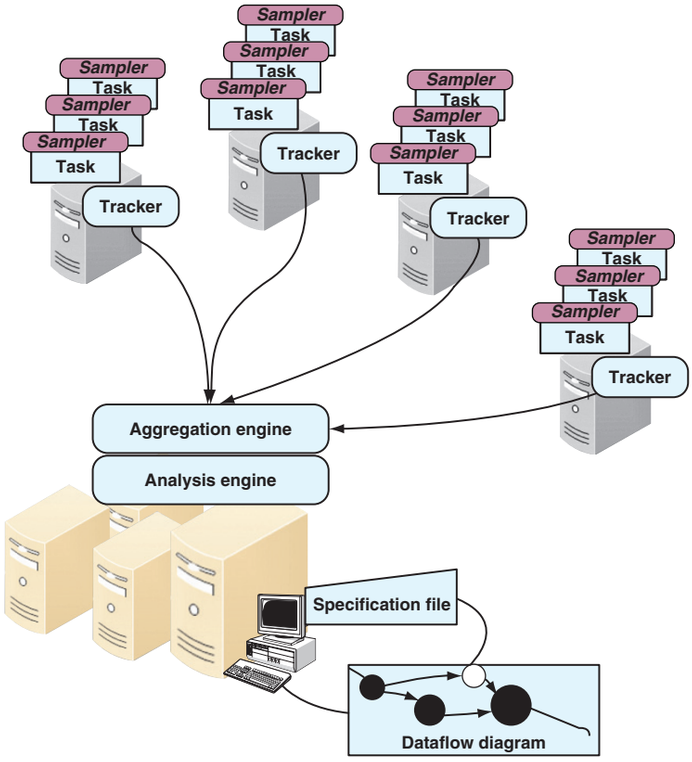

Figure 8: HiTune performance analysis framework (Source: Intel Corporation, 2012)

HiTune Implementations

Based on the Hadoop dataflow model described earlier, we have built HiHune[20], a Hadoop performance analyzer that allows users to understand the runtime behaviors of Hadoop applications so that they can make educated decisions regarding how to improve the efficiency of these massively distributed systems—just as traditional performance analyzers like gprof[21] and Intel® VTune™[22] allow users to do for a single execution of a single program.

Our approach relies on distributing instrumentations on each node in the Hadoop cluster and then aggregating all the instrumentation results for dataflow-based analysis. The performance analysis framework consists of three major components, namely the tracker, the aggregation engine, and the analysis engine, as illustrated in Figure 8.

Figure 9: Format of the sampling record (Source: Intel Corporation, 2012)

The tracker is a lightweight agent running on every node. Each tracker has several samplers, which inspect the runtime information of the programs and system running on the local node (either periodically or based on specific events), and sends the sampling records to the aggregation engine. Each sampling record is of the format shown in Figure 9.

- *Timestamp* is the sampling time for each record.

- *Type* specifies the type of the sampling record (such as CPU cycles, disk bandwidth, and log files).
- *Target* specifies the source of the sampling record. It contains the name of the local node, as well as other sampler-specific information (such as CPUID, network interface name, or log file name).
- *Value* contains the detailed sampling information of this record (such as CPU load, network bandwidth utilization, or a line/record in the log file).

The aggregation engine is responsible for collecting the sampling information from all the trackers in a distributed fashion and storing the sampling information in a separate monitoring cluster for analysis. Any distributed log collection tools (examples include Chukwa[23][24], Scribe[25], and Flume[26]) can be used as the aggregation engine. In addition, the analysis engine runs on the monitoring cluster, and is responsible for conducting the performance analysis and generating the analysis report, using the collected sampling information based on the Hadoop dataflow model.

Experience

HiTune has been used intensively inside Intel for Hadoop performance analysis and tuning. In this section, we share our experience on how we use HiTune to efficiently conduct performance analysis and tuning for Hadoop, demonstrating the benefits of dataflow-based analysis and the limitations of existing approaches, including system statistics, Hadoop logs and metrics, and traditional profiling.

Tuning Hadoop Framework

One performance issue we encountered was extremely low system utilization when sorting many small files (3200 500-KB-sized files) using Hadoop 0.20.1; system statistics collected by the cluster monitoring tools (such as Ganglia[27])

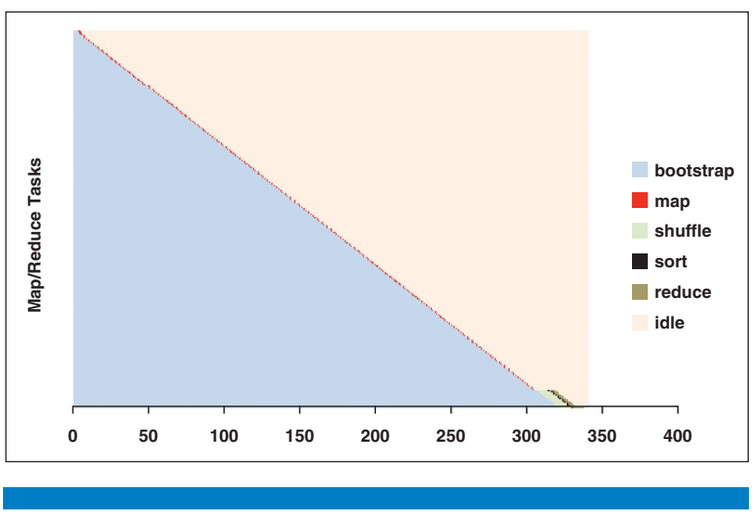

Figure 10: Dataflow execution for sorting many small files (Source: Intel Corporation, 2012)

showed that the CPU, disk I/O, and network bandwidth utilizations were all below 5 percent. That is, there were no obvious bottlenecks or hotspots in our cluster; consequently, traditional tools like system monitors and program profilers failed to reveal the root cause.

To address this performance issue, we used HiTune to reconstruct the dataflow execution process of this Hadoop job, as illustrated in Figure 10. As is obvious in the dataflow execution, there are few parallelisms between the map tasks, or between the map tasks and reduce tasks in this job. Clearly, the task scheduler in Hadoop 0.20.1 (Fair Scheduler[28] is used in our cluster) failed to launch all the tasks as soon as possible in this case. Once the problem was isolated, we quickly identified the root cause: by default, the Fair Scheduler in Hadoop 0.20.1 only assigns one task to a slave at each heartbeat (that is, the periodical keep-alive message between the master and slaves), and it schedules map tasks first whenever possible; in our job, each map task processed a small file and completed very fast (faster than the heartbeat interval), and consequently each slave ran the map tasks sequentially and the reduce tasks were scheduled after all the map tasks were done.

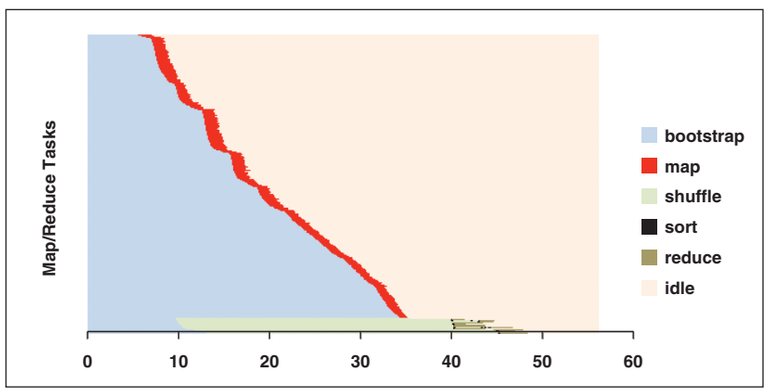

Figure 11: Sorting many small files with Fair Scheduler 2.0 (Source: Intel Corporation, 2012)

To fix this performance issue, we upgraded the cluster to Fair Scheduler 2.0[29][30], which by default schedules multiple tasks (including reduce tasks) in each heartbeat; consequently the job runs about six times faster (as shown in Figure 11) and the cluster utilization is greatly improved.

Analyzing Application Hotspots

In the previous section, we demonstrated that the high level dataflow execution process of a Hadoop job helps users to understand the dynamic task scheduling and assignment of the Hadoop framework. In this section, we show that the dataflow execution process helps users to identify the data shuffle gaps between map and reduce, and that relating the low level performance activities to the high level dataflow model allows users to conduct fine-grained, dataflow-based hotspot breakdown (so as to understand the hotspots of the massively distributed applications).

Figure 12 shows the runtime behavior of TeraSort. The dataflow execution process of TeraSort shows that there is a large gap (about 15 percent of the total job running time) between the end of map tasks and the end of shuffle phases. According to Hadoop dataflow model (see Figure 7), shuffle phases need to fetch the output from all the map tasks in the copier stages, and ideally should complete as soon as all the map tasks complete. Unfortunately, traditional tools or Hadoop logs fail to reveal the root cause of the large gap, because during that period, none of the CPU, disk I/O, and network bandwidth are bottlenecked; the "Shuffle Fetchers Busy Percent" metric reported by the Hadoop framework is always 100 percent, while increasing the number of copier threads does not improve the utilization or performance.

To address this issue, we used HiTune to conduct hotspot breakdown of the shuffle phases, which is possible because HiTune has associated all the low level sampling records with the high level dataflow execution of the Hadoop job. The dataflow-based hotspot breakdown (see Figure 13) shows

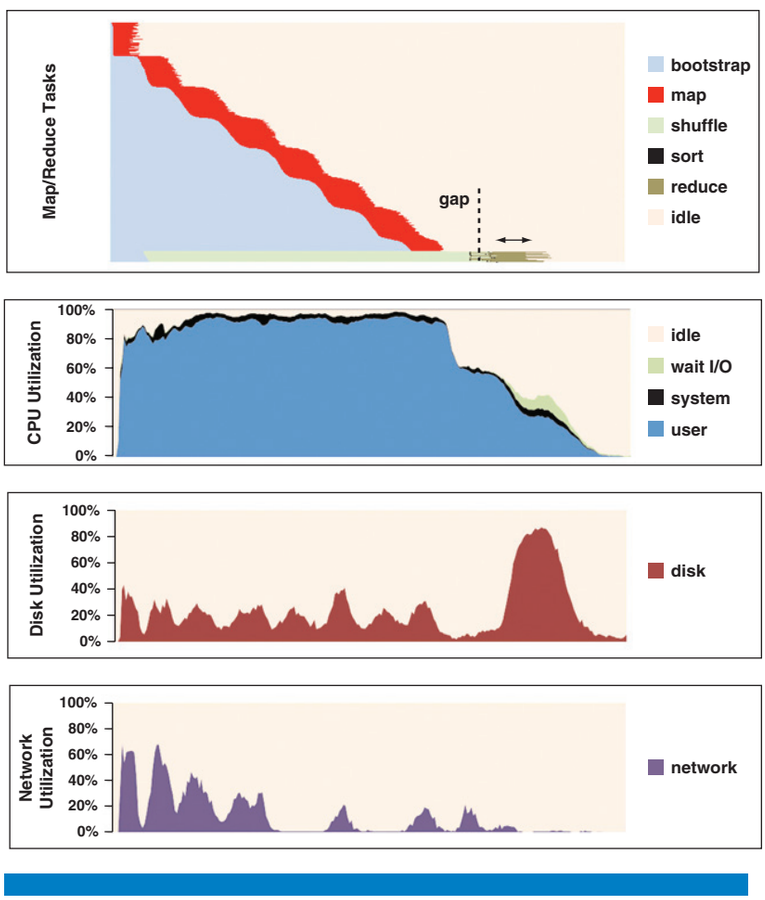

Figure 12: TeraSort (using default compression codec) (Source: Intel Corporation, 2012)

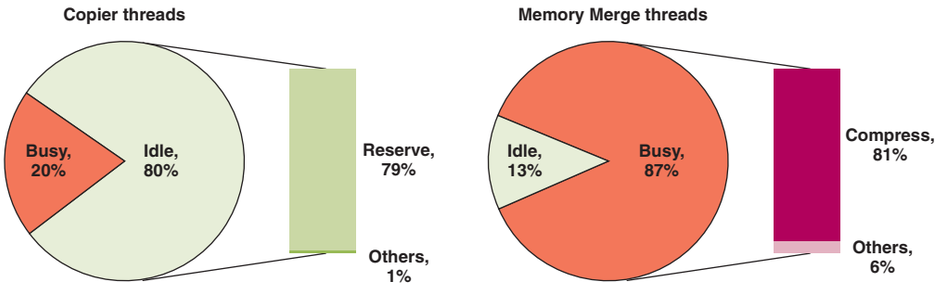

Figure 13: Copier and Memory Merge threads breakdown (using default compression codec) (Source: Intel Corporation, 2012)

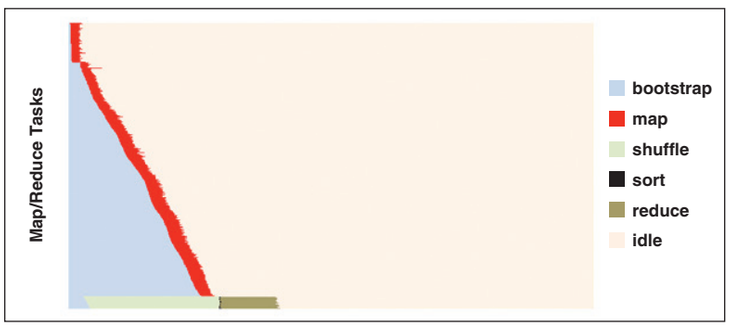

Figure 14: TeraSort (using LZO compression) (Source: Intel Corporation, 2012)

that, in the shuffle stages, the copier threads are actually idle 80 percent of the time, waiting (in the *ShuffleRamManager.reserve* method) for the occupied memory buffers to be freed by the memory merge threads. (The idle-versus-busy breakdown and the method hotspot are determined using the Java thread state and stack trace in the task execution sampling records respectively.) On the other hand, most of the busy time of the memory merge thread is due to the compression, which is the root cause of the large gap between map and shuffle. To fix this issue and reduce the compression hotspots, we changed the compression codec to LZO[31], which improves the TeraSort performance by more than 2x and completely eliminates the gap (see Figure 14).

Big Data Solution Case Study

Having moved beyond its origins in large Web sites, Hadoop is now heading toward ubiquity for big data storage and processing, including telecom, smart city, financial service, and bioinformatics. In this section, we describe the challenges in Hadoop-based solutions for these general big data applications

and illustrate how these challenges can be addressed through a case study of intelligent transportation system (ITS) applications.

Challenges for Big Data Applications

Hadoop has been emphasizing on high scalability (that is, scaling out to more servers) via linear scalability and automatic fault tolerance, as opposed to high (per-server) efficiency such as low latency and high performance. Unfortunately, for many enterprise users, it is almost impossible to set up and maintain a cluster of thousands of nodes, which negatively impacts the total cost of ownership (TCO) of Hadoop solutions.

In addition, the MapReduce dataflow model assumes data parallelism in the input applications, a natural fit for most of the big data applications. On the other hand, for some application domains, such as computational fluid dynamics, matrix computation and graph analysis in weather forecast and oil detection, and iterative machine learning, extensions to the MapReduce model are required to support these applications more efficiently.

Third, some mission-critical enterprise applications, such as those in telecommunications and financial services, require extremely high availability, with complete eliminations of SPOFs (single point of failures) and sub-second failover. Much effort will be required to improve the high availability support in the Hadoop stack.

Finally, for many enterprise big data applications, advanced security support (including authentication, authorization, access control, data encryption, and secure data transfer) is needed. In particular, fine-grained access control support is required for all components in the Hadoop stack.

Case Study: Intelligent Transportation System

In this section, we present a case study of an intelligent transportation system (ITS), a key application in smart cities, which monitors the traffic in a city and provides real-time traffic information. In the city, tens of thousands of cameras and sensors are installed in the road to capture the traffic information, including vehicle speed, vehicle identifications, and images of the vehicles. The system stores this data in real time, and users and applications need to query the data in near real time, typically scanning records within a specific time range.

HBase is a good fit for this application, as it provides real-time data ingestion and query, and high throughput table scanning. However, an ITS application is usually a large-scale distributed system, where cameras and sensors are scattered around the city, and there are edge data centers that connect to these devices through dedicated networks. Existing big data solutions (including HBase) are not designed for this type of geographically distributed data centers and cannot properly handle the associated challenges, especially the slow and unreliable WAN (wide area network) connections among different data centers.

To address this challenge, we extend HBase so that a global table can be overplayed on top of multiple HBase systems across different data centers,

which provide low latency locality-aware data capturing, automatic failover across different data centers, and a location-independent global view of the application to query the data.

Conclusions

Big data is rich with promise, but equally rife with challenges. It extends beyond traditional structured (or relational) data, including unstructured data of all varieties (text, image, video, and more); it is not only large in size, but also growing faster than Moore's law (more than doubling every two years). The massive scale, exponential growth and variable nature of big data necessitate a new processing paradigm.

The Hadoop stack has emerged as the de facto standard for big data storage and processing. It originates from big web sites and is now heading toward ubiquity for large-scale data processing, including telecom, smart city, financial service, and bioinformatics. As Hadoop-based big data solutions grow in pervasiveness and scale, efficiently tuning and optimizations of Hadoop deployments, and extending Hadoop to support general big data applications become critically important. Intel has provided many proven methodologies, tools, a solution stack, and reference architecture to address these challenges.

References

[1] McKinsey Global Institute. "Big data: The next frontier for innovation, competition, and productivity," May 2011.

[2] IDC. "Extracting Value from Chaos," June 2011.

[3] Teradata. "Challenges of Handling Big Data," July 2011.

[4] J. Dean, S. Ghemawat. "MapReduce: Simplified Data Processing on Large Clusters." The 6th Symposium on Operating Systems Design and Implementation, 2004.

[5] Hadoop. http://hadoop.apache.org/

[6] S. Ghemawat, H. Gobioff, and S. Leung. "The Google File System," 19th ACM Symposium on Operating Systems Principles, October 2003.

[7] A. Thusoo, R. Murthy, J. S. Sarma, Z. Shao, N. Jain, P. Chakka, S. Anthony, H. Liu, N. Zhang. "Hive - A Petabyte Scale Data Warehousing Using Hadoop." The 26th IEEE International Conference on Data Engineering, 2010.

[8] Hive. ***http://hadoop.apache.org/hive/***

[9] C. Olston, B. Reed, U. Srivastava, R. Kumar, A. Tomkins. "Pig latin: a not-so-foreign language for data processing." The 34th ACM SIGMOD inter-national conference on Management of data, 2008.

[10] Pig. ***http:///hadoop.apache.org/pig/***

[11] F. Chang, J. Dean, S. Ghemawat, W. Hsieh, D. Wallach, M. Burrows, T. Chandra, A. Fikes, R. Gruber. "Bigtable: A Distributed Storage System for Structured Data." 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI'06), 2006.

[12] HBase. ***http://hadoop.apache.org/hbase/***

[13] "Hadoop and HBase at RIPE NCC." ***http://www. cloudera.com/blog/2010/11/hadoop-and-hbase-at-ripe-ncc/***

[14] A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. DeWitt, S. Madden, M. Stonebraker. "A comparison of approaches to large-scale data analysis." The 35th SIGMOD international conference on Management of data, 2009.

[15] S. Huang, J. Huang, J. Dai, T. Xie, B. Huang. "The HiBench Benchmark suite: Characterization of the MapReduce-Based Data Analysis." IEEE 26th International Conference on Data Engineering Workshops, 2010.

[16] Sort Benchmark. ***http://sortbenchmark.org/***

[17] Nutch. ***http://lucene.apache.org/nutch/***

[18] Mahout. ***http://lucene.apache.org/mahout/***

[19] "A Benchmark for Hive, PIG and Hadoop," ***http://issues.apache.org/jira/browse/HIVE-396***

[20] J. Dai, J. Huang, S. Huang, B. Huang, Y. Liu. "HiTune: dataflow-based performance analysis for big data cloud," 2011 USENIX conference on USENIX annual technical conference (USENIX ATC'11), June 2011.

[21] S. L. Graham, P. B. Kessler, M. K. Mckusick. "Gprof: A call graph execution profiler." The 1982 ACM SIGPLAN Symposium on Compiler Construction, 1982.

[22] Intel VTune Performance Analyzer. ***http://software. intel.com/en-us/intel-vtune/***

[23] A. Rabkin, R. H. Katz. "Chukwa: A system for reliable large-scale log collection," Technical Report UCB/EECS-2010-25, UC Berkeley, 2010.

[24] Chukwa/ ***http://incubator.apache.org/chukwa/***

[25] Scribe. ***http://github.com/facebook/scribe***

[26] Flume. ***http://incubator.apache.org/flume/***

[27] Ganglia. ***http://ganglia.sourceforge.net/***

[28] A fair sharing job scheduler. ***https://issues.apache. org/jira/browse/HADOOP-3746***

[29] M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, I. Stoica. "Job Scheduling for Multi-User MapReduce Clusters." Technical Report UCB/EECS-2009-55, UC Berkeley, 2009.

[30] Hadoop Fair Scheduler Design Document. ***https://svn.apache.org/repos/asf/hadoop/mapreduce/trunk/src/contrib/fairscheduler/designdoc/fair_scheduler_design_doc.pdf***

[31] Hadoop LZO patch. ***http://github.com/kevinweil/hadoop-lzo***

Authors' Biographies

Jinquan (Jason) Dai is a principal engineer in Intel's Software and Services Group, managing the engineering team for building and optimizing big data platforms (e.g., Hadoop). Before that, he was the lead architect and the engineering manager for building the first commercial auto-partitioning and parallelizing compiler for many-core many-thread processors (Intel Network Processor) in the industry. He received his Master's degree from the National University of Singapore, and his Bachelor's degree from Fudan University, both in computer science.

Jie (Grace) Huang is a senior software engineer in Intel's Software and Services Group, focusing on the tuning and optimization of big data platforms. She joined Intel in 2008, after getting her Master's degree in Image Processing and Pattern Recognitions from Shanghai Jiaotong University.

Shengsheng Huang is a senior software engineer in Intel's Software and Services Group, focusing on big data platforms, particularly the Hadoop stack. She got her Master's and Bachelor's degrees in Engineering from Zhejiang University in 2006 and 2004 respectively.

Yan Liu is a software engineer in Intel's Software and Services Group, focusing on Hadoop workloads and optimizations. She holds a Bachelor's degree in Software Engineering and a Master's degree in Computer Software and Theory.

Yuanhao Sun is a software architect in Intel's Software and Services Group, leading the efforts on big data and Hadoop related product development. Before that, Yuanhao was responsible for the research and development on XML acceleration technology. Yuanhao received his Bachelor's and Master's degrees from Nanjing University, both in computer science.