Panthera: Caching and Cache-based Scheduling in Distributed Computing Systems

Dhaivat Pandya
Appleton North High School

Abstract

The adoption of distributed computing systems has grown massively in the past few years. In particular, Apache Hadoop, which allows developers to create applications that run on a cluster of computers, is currently used throughout academia and industry in areas such as machine learning, medical diagnosis, natural language processing, etc. However, the Hadoop File System fails to effectively utilize random access memory (RAM) and local storage in order to reduce waiting time (i.e. latency). In this project, named *Panthera*, caching and scheduling systems for Hadoop were developed. *Panthera* caches both information about files and the file contents, thereby reducing waiting time in downloading and accessing the files and related information. It runs with an unmodified version of Hadoop, meaning it can integrate easily into existing architecture. The results showed that data access latency was 8.63 times lower with *Panthera* and metadata access latency was 8.13 times lower. Such drastic decreases in latency can greatly increase the efficiency of existing Hadoop applications and also allow the creation of algorithms that were previously not feasible on Hadoop. *Panthera* was tested with existing Hadoop algorithms and running times were up to 3.31 times lower than the control group of a standard Hadoop installation. In addition to the caching system, a scheduler and scheduling algorithm for Hadoop were developed. They use information available about the caches to decide the order of execution for computational jobs, thereby further reducing running time. Finally, *Panthera* is widely applicable and can significantly speed up research in a very large range of fields ranging from bioinformatics to artificial intelligence. All code developed during this project will be open sourced to further development in the area.

1 Introduction

As computational problems and their associated datasets have grown in size, it has become necessary to take advantage of distributed computing systems to solve them [20]. Such systems allow programs to scale from one computer (i.e. one *node*) to thousands of computers with few or no changes in the codebase. In particular, the Hadoop distributed system [18] has seen tremendous growth. Hadoop is an open source implementation of the revolutionary MapReduce system developed at Google [5]. With

it, developers can easily take advantage of large, multi-node clusters to solve computational problems.

Hadoop consists of a computational layer termed *MapReduce* and a custom distributed filesystem named *Hadoop Distributed File System (HDFS)*. HDFS is used to allow all nodes in a cluster access to the same dataset. For instance, one may consider running an algorithm that counts the frequency of each word in a large text file. If this algorithm is run on a Hadoop cluster, each node must have access to the file in question in order to determine word frequency.

Current Hadoop deployments exist in the thousands of nodes at companies such as Cloudera, Facebook, Yahoo, etc. Especially in such large networks, *access latency*, or the waiting time associated with accessing files and related information through the Hadoop Distributed File System (*HDFS*), is a significant concern [19]. Latency in distributed systems can have significant effects and a reduction of the same comes with tremendous benefits. As the RAMCloud project has outlined [15], low latency can greatly extend the applications of distributed computing systems. For example, algorithms that require files other than the input file are largely impractical in high latency systems such as Hadoop [15, 25]. A similar case exists with jobs (or sets of jobs) that consist of multiple passes over the same dataset; many of these exist in the field of machine learning, analytics, etc. [25]. However, with lowered latency, such algorithms may become useful and applicable. HDFS currently does not use local (i.e. on the client) random access memory (RAM) to improve latency.

I present *Panthera*, a *cache layer* and *cache-based scheduling system* for Hadoop. The *Panthera* cache layer functions by retaining recently accessed files and related information in random access memory. Thus, when the files are needed once again, they do not have to be downloaded or read from the hard drive, thereby greatly reducing file access latency. Another portion of the project, the *Panthera* scheduling system utilizes information about the data blocks cached in a cluster at a given time to schedule jobs. By doing so, it aims to maximize the benefit of caching to a series of jobs.

In this paper, the *Panthera* caching layer and scheduling system are presented. In Section 2, motivation and construction of the *Panthera* architecture are explained. Section 3 describes the caching layer portion of this project. Section 4 presents the scheduling algorithm and implementation details. The results and data are noted in Section 5. Finally, Sections 6, 7, 8 conclude with an interpretation of the results, discussion of potential applications and future work.

2 Architecture

2.1 Design Constraints

The primary goal of the *Panthera* project is to reduce access latency in HDFS. Therefore, data and metadata request latency should be significantly lowered with *Panthera* in comparison to the control group of a standard Hadoop installation.

There are many systems built on Hadoop that are in widespread use [6, 1, 14]. The Hadoop project is also supported by large corporations with a consistent release cycle. Thus, for *Panthera* to be practical, it must opedrate independently of the existing

Hadoop codebase, i.e. as a layer, rather than modification. This implies that *Panthera* must implemented without a single change in the existing Hadoop codebase, i.e. as a "drop-in" solution. This is due to the fact that a deviation from the mainline Hadoop codebase would have to updated at the same pace as the vanilla Hadoop, which would be contrary the benefits of Hadoop's large and active development community. Also, since *Panthera* does not require any code modifications to existing Hadoop applications, allowing it to be easily integrated into current software stacks.

In order to develop a cache-based scheduling system, the *Panthera* caching layer must be able to report information (e.g. blocks currently cached) over the network. The cache-based scheduler then uses information about blocks cached and the files each job requires to run in order to optimize the running order of a set of jobs (see Section 4.3 for a complete description of the *Panthera* scheduling algorithm).

2.2 Metadata and Data

Within the Hadoop File System (HDFS), there is a clear architectural distinction between *metadata* and *data* [19]. Metadata refers to information about a file or a directory, e.g. file size, file names, the location of a file in the cluster, etc. For HDFS, the NameNode server handles all metadata. For every request, the NameNode recomputes metadata. *Panthera* can reduce this computation by caching the results of a previous request, thereby conserving resources.

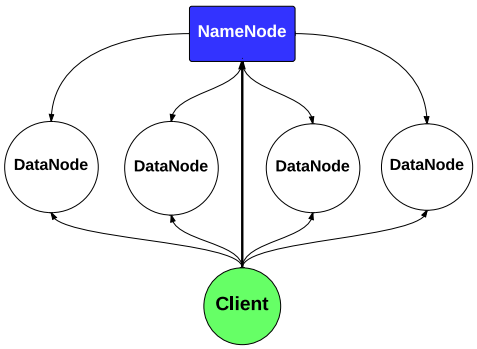
Data refers to the actual content of a file. Within HDFS, each file is split into 64 MB large *blocks*, which are stored on DataNodes. Here, *Panthera* serves to reduce access time by holding recently used blocks in RAM locally thereby eliminating the need to download files from a remote server if they are present in the cache.

Metadata and data caching provide significantly different technical challenges as the former involves conversion of computational latency to memory latency, whereas the latter involves a conversion of network and hard drive latency to memory latency. Note: memory latency is of the smallest magnitude of all three in consideration. In many cases [15] it is 1/1000th hard drive latency.

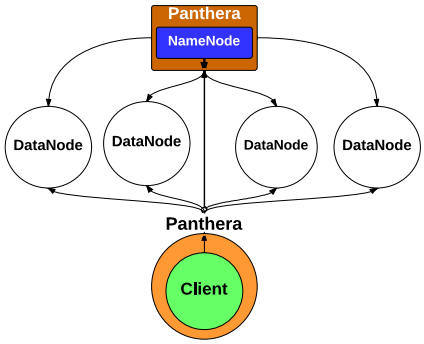
2.3 Panthera Architecture

Figure 2.3 outlines the architecture differences between a standard Hadoop File System architecture and an installation with *Panthera*. In HDFS, the *NameNode* serves as a metadata server, handling all requests from clients relating to metadata (e.g. directory listing, file sizes, block locations, etc.). *DataNode(s)* hold data and serve it at request from the client. In order to read or write a file, a client first obtains permission from the NameNode and subsequently communicates with a DataNode to complete the operation. Though the aforementioned background is sufficient for this paper, a more complete description of the Hadoop File System may be found in [18].

Panthera operates by intercepting requests and responses at the NameNode and the client. Both nodes have a running instance of *Panthera*. Every request issued from the client is first reviewed by *Panthera*, and if possible, answered immediately from the cache. If the response is not found in the cache, certain requests are cached and



(a) Vanilla Hadoop architecture



(b) Panthera-Hadoop architecture

subsequent responses are associated with the request in consideration.

The *Panthera* instance on the NameNode serves as a cache refresh mechanism. Though sparse in the Hadoop environment since input files are usually not written to in the MapReduce computational paradigm, files that are cached may become out-of-date, i.e. the cache becomes "stale". The *Panthera* instance on the NameNode monitors the filesystem for changes and reports certain changes to Panthera caches, which then remove stale files from the cache. The refresh mechanism is particularly important for metadata caching where the cached information is far more likely to change during the execution of a job.

The *Panthera* cache layer, since it caches both data and metadata, can be considered two separate subsystems, which are now discussed.

3 Panthera Caching Layer

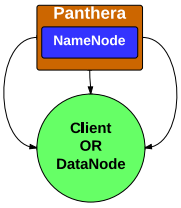
A significant portion of Panthera is a caching layer for Hadoop, the details of which are now covered.

3.1 Metadata caching

The *Panthera* metadata system runs on the NameNode and communicates with the *Panthera* instance on the client. The challenge on the metadata portion consists of converting computational latency to memory latency. For example, for some intensive methods (e.g. a recursive directory listing), it may be worthwhile to cache a response in order to prevent computing the same list again.

However, the problem lies in identifying such methods. Obviously, there are some methods for which caching is not worthwhile since their execution time is lower or comparable to memory latency. Within Hadoop, there are several methods that incur very significant computational latency, e.g. a recursive directory listing. Only these methods are cached within Panthera.

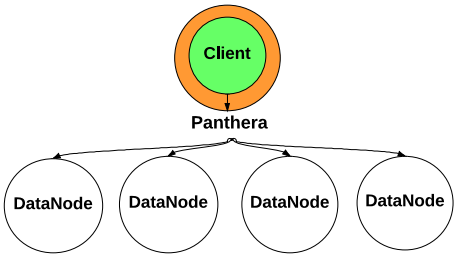
Figure 2: *Panthera* Metadata System



Currently, a standard LRU cache [22] is used for each method. Testing of different caching methods and predictive prefetching algorithms [**?**] is under way.

3.2 Data caching

Figure 3: *Panthera* Data System



Panthera's data caching system runs on the client. The architecture is similar to that of the metadata caching system. Every request that the client makes to the DataNodes is first processed by *Panthera* and responded to from the cache if possible.

The challenges with data caching differ significantly from those in metadata caching. Here, the goal is to convert network and hard drive read latency into memory latency. As [15, 18] note, memory latency can be extremely low in modern systems, especially in comparison to network and hard drive latencies. Thus, in some respects, the goal of lowered latency is "easier" to attain in the data cache in comparison to the metadata cache.

4 Scheduler

4.1 Hadoop Job Structure

Hadoop, in professional and academic environments, is often used with series rather than individual jobs. For example, in machine learning, various algorithms (termed

iterative machine learning algorithms) consist of multiple Hadoop jobs rather than a single job [25]. In Hadoop-based querying systems [**?** 14], a set of queries may consist of jobs that are entirely independent of one-another. Deciding how to order these jobs depending on cached file availability is a problem of great interest; *Panthera* presents the solution of a cache-based scheduler for Hadoop.

4.2 Cluster Saturation

It may be asked why it is not possible to run independent jobs in parallel on the Hadoop cluster. Each node in a given cluster has a maximum number of *map* or *reduce* calls (i.e. the basic computational units in Hadoop) it may process at a given time. If a job saturates nodes in a cluster, running another job concurrently would provide no running time benefit. Thus, a scheduler is useful since it may organize independent jobs in a beneficial ordering. In *Panthera*, this ordering is based on the files or blocks present in the cluster's caches.

4.3 Scheduling Algorithm

The scheduling problem faced may be stated as:

> Given $\mathbf{J} = J_1, J_2, ..., J_k$ jobs to be run on a Hadoop cluster with caches $\mathbf{C} = C_1, C_2, ..., C_n$, what ordering will best take advantage of the caches?

Each J_i accesses a set of blocks F_i and each C_i holds a set of blocks B_i. *Panthera* assigns every J_i a *job score*, denoted x_i:

$$x_i = \frac{|F_i \cap B_t|}{|B_t|} \text{ where } B_t = \bigcup_{j=1}^{n} B_j \tag{1}$$

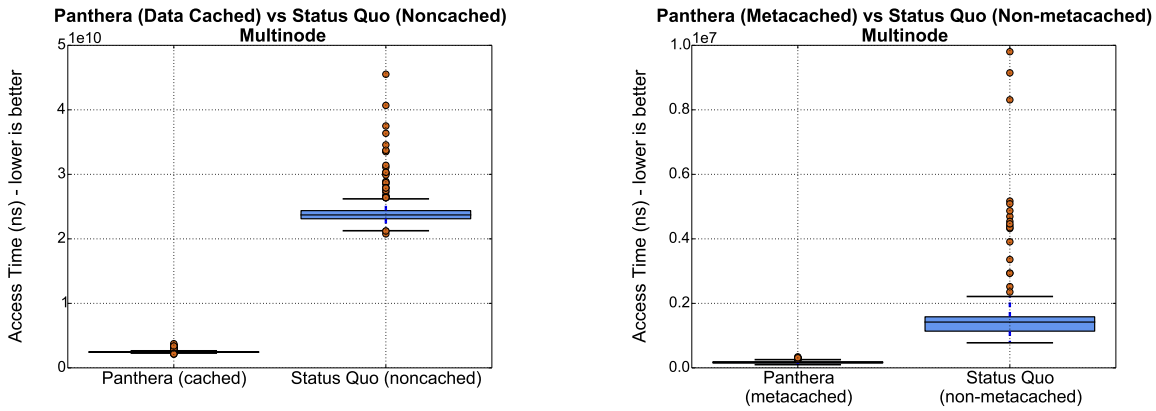
The *Panthera scheduling algorithm* consists of sorting $\mathbf{J}$ in descending order with respect to the x_i associated with each J_i.

While J_i is running, *Panthera* may prefetch data for J_{i+1} to further reduce latency. An implementation of predictive prefetching algorithms as described in [7] is possible in *Panthera* and may serve to further reduce latency in specific applications.

5 Results

5.1 Testing Methodology

In order to test the metadata cache, a directory listing query was repeatedly run on a directory with 100 files in it, and the latency times for responding are measured, with *Panthera* and without. The reasoning for picking such a test lies in the fact that a directory listing in Hadoop involves all six of the cached methods as well as a few non-cached methods. Thus, the test provides a good estimate of the efficacy of the cache. Latency times for the data cache were obtained by repeatedly querying for a 64 megabyte large file. The file size was chosen since it is the default block size for a



(a) Data access latency

(b) RefCount job performance

Figure 4: Metadata access latency

Type	Mean Time Improvement	Standard Dev. Contraction
Metadata	8.13x	27.38x
Data	8.63x	11.14x
WordCount	1.25x	1.02x
Ref. count	3.31x	22.91x

Table 1: Latency improvement with *Panthera*

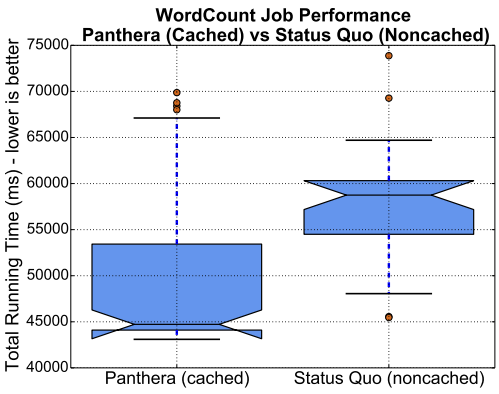
Hadoop cluster and the *Panthera* cache stores files as blocks. Both metadata and data cache testing was conducted on virtualized server instances running on hardware with solid state hard drives.

In order to test the effects of *Panthera* on the Hadoop computing system (rather than focusing on only HDFS access time), running times of two existing algorithms were recorded. The first is *WordCount*, the classic distributed algorithm that counts the number of occurrences of each word in a text file. The second has been, termed *RefCount*, has thus far been impractical on Hadoop due to excessive running times. It is similar to WordCount in that it counts occurrences of words, but it consults a *reference file* (e.g. a limited dictionary) to determine which words are to be counted.

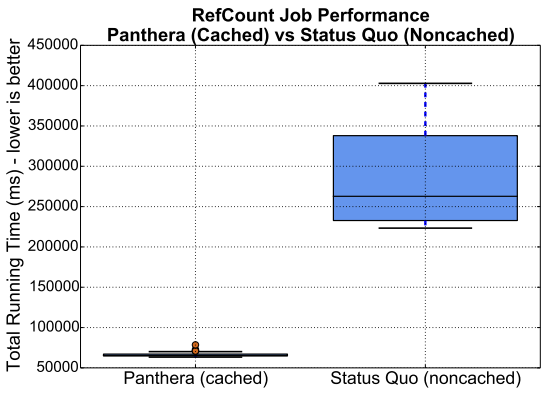
5.2 Results

The *Panthera* data cache decreased file access latency by a factor of 8.63 and metadata access latency was lowered by a factor of 8.13, both representing drastic decreases in HDFS latency. In addition, the standard deviation was also significantly decreased with *Panthera*: by a factor 27.38 for metadata and a factor of 11.14 for data. With a lower standard deviation, the predictability of performance for Hadoop applications increases.

The tests with existing Hadoop algorithms show similarly positive results. *Panthera*



(a) WordCount job performance



(b) RefCount job performance

Figure 5: Algorithm running time comparison

is able to reduce running time for the WordCount algorithm by 25% and is able to do the same for the Ref. Count algorithm by 331%. It is important to keep in mind that these performance boosts were attained with the original algorithms left entirely unmodified. The difference between the two lies in the fact that WordCount has little interaction with HDFS once the initial input files have been downloaded, whereas Ref. Count often obtains the reference file from HDFS. Notice that the testing was conducted with only one slave node. Adding slave nodes to a cluster would increase the number of file requests since each node requires a portion of the input file.

6 Conclusions

Panthera fulfils all of the constraints mentioned at the outset. It is a layer on Hadoop rather than a main repository code modification, allowing existing Hadoop applications to remain entirely unmodified. Considering the latency measurements, it is clear that it has also met the requirements in terms of benefit. *Panthera* was able to reduce data access latency by a factor of 8.63 and reduce basic metadata access latency by 8.13. Also, *Panthera* was able to minimize the spread of latency times in both data and metadata requests. Thus, *Panthera* opens up great possibilities within Hadoop due to the fact that algorithms that were previously impractical (e.g. distributed tree traversal algorithms) may now be implemented. Additionally, a scheduler algorithm was devised and a cache-based scheduler was implemented for Hadoop which enables further reduction of latency via greater utilization of cached resources.

Previous work [19, 2] has lamented the latency issues that have plagued HDFS and Hadoop as barriers to adoption. *Panthera* significantly reduces the scope of these issues and allows the application of Hadoop in areas where its reach has been limited.

7 Applications

One of the major benefits of the systems developed is that they are independent of the existing Hadoop codebase and they do not require modifications to existing Hadoop applications. Thus, *Panthera* may be deployed in a wide range of scenarios, such as the following:

- Bioinformatics: Hadoop is currently in use in bioinformatics research [20]. Panthera can be employed to reduce access latencies of various datasets, e.g. protein data banks, etc. Additionally, domain-specific prefetching can be performed to further improve performance.
- Machine learning: Apache Mahout is a machine learning library for Hadoop. Panthera has been tested with the K-Means implementation it provides and there is evidence of significant performance improvement.
- Predictive analytics: A quickly growing field that has thoroughly embraced Hadoop [3]. Metadata caching can be used to decrease human operator waiting time. Often, queries consist of several queued jobs, meaning that the *Panthera* scheduler can be used to optimize job ordering.

8 Future work

One of the most immediate avenues for further research is in *domain-specific prefetching*. Different problem domains (e.g. natural language processing, predictive analytics, etc.) have different dataset requirements. It may be possible to predict what files a given job will need depending on what problem domain the job is in. A generalized version of the same problem would be *predictive prefetching* [7] which involves the prediction and prefetching of files that will be required in the future. In both these problems, the Hadoop environment provides unique challenges.

Another area worthy of further exploration is the effect of different cache replacement algorithms in Hadoop. Though various cache replacement algorithms have been explored thoroughly at the local filesystem level, their study in conjunction with existing Hadoop algorithms may provide novel results, i.e. the access patterns of certain procedures may work best with a certain cache replacement scheme [16].

References

[1] CloudBATCH: A batch job queuing system on clouds with Hadoop and HBase, author=Zhang, Chen and De Sterck, Hans. In *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, pages 368–375. IEEE, 2010.

[2] Dhruba Borthakur, Jonathan Gray, Joydeep Sen Sarma, Kannan Muthukkaruppan, Nicolas Spiegelberg, Hairong Kuang, Karthik Ranganathan, Dmytro Molkov, Aravind Menon, Samuel Rash, et al. Apache Hadoop Goes Realtime at Facebook.

In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 1071–1080. ACM, 2011.

[3] Hsinchun Chen, Roger HL Chiang, and Veda C Storey. Business intelligence and analytics: From big data to big impact. *MIS Quarterly*, 36(4), 2012.

[4] Sudipto Das, Yannis Sismanis, Kevin S Beyer, Rainer Gemulla, Peter J Haas, and John McPherson. Ricardo: Integrating R and Hadoop. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 987–998. ACM, 2010.

[5] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[6] Lars George. *HBase: the definitive guide*. O'Reilly Media, Inc., 2011.

[7] Jim Griffioen and Randy Appleton. Reducing File System Latency using a Predictive Approach. In *USENIX Summer*, pages 197–207, 1994.

[8] Karthik Kambatla, Abhinav Pathak, and Himabindu Pucha. Towards Optimizing Hadoop Provisioning in the Cloud. In *Proc. of the First Workshop on Hot Topics in Cloud Computing*, page 118, 2009.

[9] David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin. Consistent Hashing and Random trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 654–663. ACM, 1997.

[10] Kamal Kc and Kemafor Anyanwu. Scheduling Hadoop Jobs to Meet Deadlines. In *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, pages 388–392. IEEE, 2010.

[11] Jacob Leverich and Christos Kozyrakis. On the energy (in) efficiency of hadoop clusters. *ACM SIGOPS Operating Systems Review*, 44(1):61–65, 2010.

[12] Xuhui Liu, Jizhong Han, Yunqin Zhong, Chengde Han, and Xubin He. Implementing WebGIS on Hadoop: A case study of improving small file I/O performance on HDFS. In *Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on*, pages 1–8. IEEE, 2009.

[13] Michael N Nelson, Brent B Welch, and John K Ousterhout. Caching in the Sprite Network File System. *ACM Transactions on Computer Systems (TOCS)*, 6(1):134–154, 1988.

[14] Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins. Pig latin: a not-so-foreign language for data processing. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1099–1110. ACM, 2008.

[15] John Ousterhout, Parag Agrawal, David Erickson, Christos Kozyrakis, Jacob Leverich, David Mazières, Subhasish Mitra, Aravind Narayanan, Guru Parulkar, Mendel Rosenblum, et al. The Case for RAMClouds: Scalable High-performance Storage Entirely in DRAM. *ACM SIGOPS Operating Systems Review*, 43(4):92–105, 2010.

[16] Thomas Roberts Puzak. Analysis of cache replacement-algorithms. 1985.

[17] Antony Rowstron and Peter Druschel. Storage Management and Caching in PAST, a Large-scale, Persistent Peer-to-Peer Storage Utility. In *ACM SIGOPS Operating Systems Review*, volume 35, pages 188–201. ACM, 2001.

[18] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The Hadoop Distributed File System. In *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, pages 1–10. IEEE, 2010.

[19] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The Hadoop Distributed File System. In *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, pages 1–10. IEEE, 2010.

[20] Ronald C Taylor. An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics. *BMC bioinformatics*, 11(Suppl 12):S1, 2010.

[21] An-I Wang, Peter L Reiher, Gerald J Popek, and Geoffrey H Kuenning. Conquest: Better Performance Through a Disk/Persistent-RAM Hybrid File System. In *USENIX Annual Technical Conference, General Track*, pages 15–28, 2002.

[22] Jia Wang. A Survey of Web Caching Schemes for the Internet. *ACM SIGCOMM Computer Communication Review*, 29(5):36–46, 1999.

[23] Jia Wang. A Survey of Web Caching Schemes for the Internet. *ACM SIGCOMM Computer Communication Review*, 29(5):36–46, 1999.

[24] Jianwu Wang, Daniel Crawl, and Ilkay Altintas. Kepler+ Hadoop: a General Architecture Facilitating Data-Intensive Applications in Scientific Workflow Systems. In *Proceedings of the 4th Workshop on Workflows in Support of Large-Scale Science*, page 12. ACM, 2009.

[25] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: cluster computing with working sets. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, pages 10–10, 2010.