Proposal for a Thesis in the Field of Information Technology

For Partial Fulfillment of Requirements For a Master of Liberal Arts Degree

Harvard University Extension School

Submission Date 12/16/2016

Cyril Allen

1525 9th Ave., Apt 2601

Seattle, WA 98101

cyril0allen@gmail.com (mailto:cyril0allen@gmail.com)

Tentative Thesis Title

Improving Data Placement Decisions for Heterogeneous Clustered File Systems

Abstract

Today, many applications such as MapReduce or the Nutanix file system, assume homogeneous constituents due to their random data placement strategies. Not all machines are equal, so one cannot expect for MapReduce tasks to be completed at the same time, or high performance Nutanix file system writes to be as fast as they would be in a homogeneous cluster. It has been shown that having a platform which proactively balances the file system workload based on node capabilities can improve performance [5][6][7][8]. In this document, I propose an adaptive data placement technique for the Nutanix file system, aiming to improve performance in heterogeneous clusters using NDFS.

Introduction

With the advent of cloud computing, datacenters are making use of distributed applications more than ever. Companies like Google use software such as MapReduce to generate over 20 petabytes of data per day using very large numbers of commodity servers [3]. Many other companies use large scale clusters to perform various computational tasks via the the open-source MapReduce implementation, Hadoop [4], or they can possess a virtualized datacenter allowing them to migrate virtual machines between various machines for high-availability reasons. As economics change for hardware, it is likely that a scalable cloud will have the requirement to mix node types, which will lead to higher performance/capacity nodes being mixed with lower performance/capacity HDD nodes. I seek to implement an adaptive data placement algorithm in the Nutanix distributed file system which will attempt to remedy the common problems found in many heterogeneous clustered file systems. Before one can

understand the performance problems that can be found in these mixed-node clusters, they must first be given an overview of common clustered file systems.

Distributed File Systems and Data Placement Overview

Hadoop Distributed File System (HDFS)

HDFS is an open source distributed filesystem written in Java that is used by Hadoop clusters for storing large volumes of data [15]. An HDFS cluster is comprised of a single NameNode and DataNodes which respectively manage cluster metadata and store the data. HDFS stores files as a sequence of same-size blocks that are replicated across DataNodes to provide fault tolerance in the event of node failures. Decisions regarding the replication and placement of blocks of data are made by the NameNode. Since large HDFS clusters will span multiple racks, replica placement within HDFS is made in a rack-aware manner to improve data reliability and network utilization. As of HDFS version 1.2.1, there is no additional data placement intelligence.

The Nutanix Distributed File System (NDFS)

NDFS is a distributed file system created by Nutanix Inc., a San Jose based company [1]. NDFS is facilitated by a clustering of controller virtual machines (CVMs) which reside, one per node, on each server in the cluster. The CVM presents via NFS (for VMWare's ESXi [14]), SMB (for Microsoft's Hyper-V [17]), or iSCSI (for Nutanix's AHV [1]) an interface to each hypervisor that they reside on. For example, the interface provided by the CVMs to VMware's ESXi hypervisor [14] will be interfaced with as a datastore. The virtual machines' VMDK files will reside on the Nutanix datastore and be accessed via NFS through the CVM sharing a host with the user VM.

Nutanix Cluster Components

Within the CVM lies an ecosystem of processes that provide the services of the NDFS. Before I can explain the work proposed for this thesis, there are two CVM processes that must be explained in more detail.

Cassandra (Distributed Metadata Store)

Cassandra stores and manages all cluster metadata in a distributed manner. The version of Cassandra running in NDFS is a heavily modified Apache Cassandra [2]. One of the main differences between Nutanix Cassandra and Apache Cassandra is that Nutanix has implemented the Paxos [18] algorithm to enforce strict consistency.

Stargate (Data I/O Manager)

The Stargate process is responsible for all data management and I/O operations. The NFS/SMB/iSCSI interface presented to the hypervisor is also presented by Stargate. All file allocations and data replica placement decisions are made by this process.

As the Stargate process facilitates writes to physical disks, it gathers statistics for each disk such as:

- The number of operations currently in flight on the disk (queue length)
- How much data in bytes currently resides on the disk
- Average time to complete an operation on the disk

Note that these statistics are only gathered on the local disks; however, they are then stored in Cassandra along with the statistics gathered by every other Stargate in the cluster. These disk statistics stored in Cassandra are pulled periodically (currently every 30 seconds) and are then used to make decisions on data placement when performing writes.

Storage Tiering

Nutanix clusters are composed of servers that contain both SSDs and HDDs. These disks obviously have a very large performance differential, so NDFS has a notion of storage tiers. Each storage tier contains similar groupings of disks so that NDFS can migrate "cold" or unused data from a tier containing fast disks (such as SSDs or NVMe drives) down to a tier containing slower disks (such as HDDs). This frees the hot tier to store frequently used data and allows for optimizations in the file system such as a persistent write buffer that only sends random writes to SSDs and coalesces the data to before down-migrating to the HDDs via a single large and sequential write. Stargate's data placement decisions are performed on a per-tier basis.

Replica Disk Selection

When a write is made in NDFS, data is written to multiple disks. The number of data replicas is determined by a configurable setting called the Replication Factor (RF). In an RF3 cluster, Stargate will attempt to place data on a local disk and two other copies of the data that will each reside on different remote nodes. Similar to HDFS, NDFS will also attempt to place data in a rack-aware manner. This will prevent data loss scenarios if a single node or rack of nodes were to irreperably fail.

Currently, Stargate will attempt to select a number of disks corresponding to the cluster RF that satisfy the following criteria:

- No two disks may reside on the same node.
- If there is a set of disks on the local node that can house a replica, choose one of those disks.

The disk selection logic is implemented via a random selection of all disks in the storage tier. When a disk is selected, a node exclusion is asserted for subsequent candidate disks so that we do not select two disks on the same node.

This selection methodology can be problematic for heterogeneous clusters since it inherently assumes all disks in a specific tier are under similar load and that each node can drive the same amount of writes to its disks.

Thesis Project Description

Motivation

A number of scenarios arise in heterogeneous Nutanix clusters that can degrade performance for an entire cluster. The currently replica disk selection logic Stargate uses does not take into account a number of variables. For example, there can be disparities in the following:

- Tier size
- CPU strength
- Running workload
- Disk health

Considering that a write is not complete until both copies (in an RF2 cluster) are written, the write's performance is at the mercy of the slowest disk/node combination. There are several scenarios, both pathological and daily occurences, where a more robust replica placement heuristic is required. For the work in this thesis proposal, I will focus on two orthogonal cases described below.

Interfering Workloads

Suppose we have a 3-node homogeneous cluster with only 2 nodes hosting active workloads. In the current random selection scheme in use by NDFS, writes are equally likely to place their replica on the other node with an active workload as they would be to place it on the idle node. This can impact performance on both the local and remote workloads. An adaptive replica placement scheme would avoid the busy node and place its remote replica on the idle node.

Nodes with Severe Tier Disparities

Let's imagine we have a 3-node heterogeneous cluster with 2 high-end nodes and a single weak node. Suppose these high-end nodes have 500GB of SSD tier and 6TB of HDD tier and the single weak node has only 128GB of SSD tier and 1TB of HDD tier. If 3 simultaneous workloads were to generate data such that the working sets of the workloads are 50% of the local SSD tier, the weaker node is at a significant disadvantage. Given the current NDFS replica selection algorithm, we can expect 500GB of replica traffic to flood the weak node and fill up its SSD tier well before the workload is finished. An adaptive replica placement heuristic would mitigate this issue by taking disk usages into consideration.

Adaptive Data Placement

To remedy the problems that arise from hetereogeneous cluster configurations, I plan to implement an adaptive replica placement algorithm for the Nutanix distributed file system. This algorithm uses a "fitness value" that is calculated from various statistics available for each disk such as disk fullness and queue depth. These fitness values are used to rank each disk when selecting where the Stargate will place data. This should mitigate many problems seen with heterogeneous Nutanix clusters.

Fitness Values

A fitness value is a number calculated from the disk stats found in NDFS metadata store. During every NDFS disk stats update, new usage and performance stats for each disk are pulled from the Cassandra database and stored in Stargate memory at some periodic interval. For the work described in this proposal, the stats used will be disk fullness percentage (fp) and disk queue length (ql) in a function that defines fitness value as:

```
f = w1 * (fp / 100) ^ (1/2) + w2 * max(200 - ql, 0) / 200
```

The w1 and w2 variables are the weights for the disk fullness and queue length terms respectively. These can be thought of as the maximum value allowed to be contributed to the fitness value for each stat.

Metrics: Disk Queue Length

Disk queue length is the average number of Stargate operations in flight for the duration of the last round of gathered statistics. A value of 200 operations was chosen as the ceiling; therefore, anything higher than this ceiling will not contribute to the fitness value. I've chosen to use this metric in the fitness function due to the fact that it can be used across different types of drive technologies without needing to change the default "worst-case" values used when there is a failure gathering disk stats. If I were to use average times for op completion, there would be a need to consider the type of drive whose fitness is being measured. This metric combats the interfering workload problem seen in both heterogeneous and homogeneous clusters.

Metrics: Disk Fullness

Disk fullness percentages are used in the fitness calculation so that disk utilization remains mostly uniform as workloads generate data. Some heterogeneous clusters include storage-heavy nodes with many large capacity disks alongside nodes with low storage capcity. This metric combats the tier disparity problem seen in heterogeneous clusters.

Since fitness values are used as weights during disk selection, an exponential decay function was chosen for the disk fullness term because it provides a property such that if two disks have some usage percentage differential, there is a consistent relationship between the selection probabilities. For example, the difference in selection probability for two disks that have a usage percentage of 10% and 20% will be the same for two disks that have a usage percentage of 50% and 60%.

Weighted Random Selection Algorithms

Part of this work will require an exploration of various weighted random selection algorithms that allow for a weighted "N choose 2" and an analysis of their behavior and performance under various workloads and conditions. After a weight is calculated for a disk in the cluster that will store a replica, we will perform a weighted random selection on the set of potential candidate disks. The schemes I will explore in this work are summarized in the next section.

Upon implementation of various selection algorithms, it will be necessary to evaluate the chances of encountering their failure scenarios in day-to-day file system operations and

realistic workloads.

Trucation Selection

Trucation selection was first introduced by Muhlenbein and Schlierkamp-Voosen in 1993 as part of their work on Breeder Genetic Algorithms [9]. The basic idea is that there is a threshold percentage (T) that will indicate the top most T% fit elements in a set. From this top T%, individuals are selected and mated randomly until the number of offspring can replace the parent population.

For the purposes of the work in this proposal, there is no need for the breeding aspect of Muhlenbein and Schlierkamp-Voosen's work. I will simply stop at the uniform random selection of individuals from the top T% elements in the set of disks. To avoid selecting the same disk from the set multiple times, it will be required that an exclusion scheme be employed. The way I will handle this is to simply remove the object from the sampling set if it is determined unsuitable for the purposes we're performing selection for. A more concrete example would be as follows, suppose we have N objects (x_1, x_2, ..., x_n) in an array (A) that represents the top T% of the total set. If we were to perform a uniform random selection from this subset and yield x_2, I would simply swap x_2 with the first element in the array x_1 and only consider the elements in A[1:]. This scheme is worst-case O(N) and guarantees that some fixed percentage of the weaker candidates will not be selected.

Stochastic Universal Sampling (SUS)

SUS is another sampling technique first introduced by Baker in 1987 [10]. The algorithm can be understood as follows:

On a standard roulette wheel there's a single pointer that indicates the winner. The roulette wheel's "bins" can all be the same size which would indicate a uniform probability of selecting any bin and could also be unevenly sized which would indicate a weighted probability. SUS uses this same concept except allows for N evenly spaced pointers corresponding to the selection of N items. Key things to note are that the set, or "bins" in my roulette analogy, must be shuffled prior to selection. Also, there is a minimum spacing allowed for the pointers to prevent selection of the same bin.

Reservoir Sampling

Reservoir sampling describes an entire family of stochastic algorithms for randomly sampling items from a very large set. Typically, the number of elements in the set is so large that it cannot fit into main memory. This summary will only focus on non-distributed reservoir sampling; therefore, the work of M.T. Chao [13] is omitted.

Algorithm R

Algorithm R was introduced by Vitter in 1985 [11] and runs in linear time proportional to the number of elements in the set/stream. The algorithm can sample k items as follows:

1. Select the first k items from the stream and insert into an array of size k.
2. For each item remaining in the stream j, such that j > k, choose a random integer M in [1, j].
3. If M <= k, replace item M of the array with item j.

Vitter showed that the probability of selecting an item after i iterations of the algorithm is k/i. This is not directly useful for my investigation into weighted random sampling; however, it is useful to give an overview of this algorithm since Efraimidis and Spirakis [12] use a variation of AlgorithmR that utilizes a random sort and weights to accomplish a weighted-random variation of reservoir sampling. This is explained in the next section.

Weighted Random Sampling via Reservoir

This algorithm performs a weighted random selection via the use of a priority queue data structure. To sample k elements from a set/stream, the algorithm is summarized using the following pseudocode:

```
While the stream has data:
  r = InclusiveRandom(0,1) ** (1 / element_weight)
  if p_queue.Size() < k:
    p_queue.Insert(r, element)
  else:
    p_queue.PopMin()
    p_queue.Insert(r, element)
```

Efraimidis and Spirakis proved that the resulting priority queue will contain a set of k elements whose probability of being included with the set is proportional to their weights.

Alternative Sampling Methods

I will also explore the two-choice method introduced by Azar et. al. [19] for and its effect on very large sets of disks. The two-choice method in this proposed work can be summarized as selecting two disks at random from the set of all possible disks and considering the higher fitness chosen disk. If we treat the NDFS replica placement problem as a balls and bins model (where the balls are data replicas and bins are nodes chosen for placement), it is known that with high probability, the maximum number of replicas placed on each node is approximately log(n)/log(log(n)). Azar et. al. showed that use of the two-choice method can reduce the maximum number of replicas placed on each node is log(log(n))/log(2).

This selection method is ideal in two scenarios:

1) When there are too many disks to perform a linear or logarithmic selection for performance reasons.

2) When one would like to avoid herding behaviors inevitably found in heavily utilized clusters that add new nodes.

Testing and Benchmarks

Testing Replica Selection via Synthetic Workload Generation

To determine the viability of various fitness function parameters in different heterogeneous cluster topologies, I will need to repeatedly simulate specific workloads on varying hardware

configurations. A tool must be written to deploy virtual machines on each node in the cluster and run a workload similar to those that can be found in real production systems.

The exact APIs used for deployment of virtual machines will vary based on the host hypervisor. The Nutanix Acropolis hypervisor exposes a REST API, which will allow the freedom in tools development to write code in almost any language. However, the Acropolis REST API is poorly documented and does not have many examples on usage. On the other hand, VMware's ESXi hypervisor exposes a Python library (PyVMomi) which allows for complete automation of virtual machine deployment and configuration. PyVMomi is very well documented and has multiple community samples, but the choice in programming language will be strictly limited to Python or Ruby.

Once a virtual machine is deployed, it must generate I/O in a configurable manner. For this, I will use fio to drive I/O on the virtual machine. This will allow for repeatability of various workloads with specific read/write ratios, queue depths, run times, and working sets. My workload generation tool will connect to each virtual machine simultaneously via SSH and run the fio script corresponding to the desired workload.

After the fio scripts have finished running, the workload generation tool must aggregate various performance statistics gathered by the fio processes running on each virtual machine. This can then be analyzed at a later time after the clusters are torn down.

Related Work

Xie et. al. showed that data placement schemes based on the computing capacities of nodes in the Hadoop Distributed File System (HDFS) significantly improved workload performance [5]. These computing capacities are determined for each node in the cluster by profiling a target application leveraging the HDFS. Their MapReduced wordcount and grep results showed up to a 33.1% reduction in response time. Similarly, Perez et. al. applied adaptive data placement features to the Expand parallel file system based on available free space [8]. Though effective in their given contexts, the main drawback to this work is that it assumes the specific application is working without interference and does not account for other workloads on the system.

One adaptive data placement approach that can account for other workloads on the system was introduced by Jin et. al. in their work on ADAPT [7]. The work predicts how failure-prone a node in a MapReduce cluster is and advises their availability-aware data placement algorithm to avoid those nodes. This proves useful for performance by avoiding faulty nodes that could fail mid-task and cause data transfers and re-calculation of data.

Work by Suresh et. al. approaches adaptive replica selection in much the same way proposed in this paper, though their work was mainly focused on decreasing tail latencies for Cassandra reads [16]. Their load balancing algorithm, C3, incorporates the concept of a value calculated from request feedback from their servers that allows for decisions to be made on server selection. In addition to the ranking function in C3, they implemented a distributed rate control mechanism to prevent scenarios where many individual clients can bombard a single highly desirable server with requests. Many of the same problems that the work in this proposal seeks to remedy are also addressed by the C3 algorithm; however, given the Nutanix file system's architecture, some C3 solutions are not feasible.

The C3 algorithm takes into account the request queue length of certain servers similar to the way I will use disk queue lengths. In addition Suresh et. al. factor in the service latencies of each server so that they may consider a different ideal queue length for each server. With this approach, longer service times will warrant a lower queue length and vice versa. This is beneficial for scenarios where there are multiple underlying storage technologies such as NVMe drives, SSDs, and HDDs under consideration, but the Nutanix file system's architecture does not allow for multiple replicas to span storage tiers. This forces the ideal queue lengths for each selection pool to be the same. Therefore, my work does not incorporate service latencies in the fitness value calculations.

Herding of requests to a single highly suitable server is a problem that arises in any replica ranking algorithm. C3 mitigates this issue by rate limiting client requests to each server via a decentralized calculation using a configured time interval and local sending rate information. I've opted to use a simpler probabilistic spreading of client requests via selecting remote nodes using a weighted random selection tied to the calculated fitness values.

Hardware Requirements

The replica placement scheme proposed earlier can be verified using an exhaustive battery of software tests. However, we still require real hardware for the performance testing of synthetic workloads in heterogeneous clusters. If we are to verify that the proposed work will not introduce a performance regression in existing configurations and also improve performance in current troublesome scenarios, we will need to create multiple different hardware configurations.

The hardware required for each problem is shown the table below:

Problem to Test	Required Hardware
Homogeneous cluster regression tests	4x identical Nutanix/Dell/Lenovo nodes
Storage-only node performance	Storage-only Nutanix nodes
Heterogeneous hybrid cluster performance	1x weak hybrid node, 2x strong hybrid node
Heterogeneous hybrid and all flash cluster performance	1x all-flash node, 2x any hybrid node

Storage tier size differentials can be simulated via modification of the Nutanix distributed file system, but if the hardware is available, it makes the most sense to use real hardware to run the automated benchmarks.

Software Requirements

All software requirements are satisfied by the Nutanix development environment. This includes:

- The unit test infrastructure for replica selection simulations. This allows for quick prototyping of schemes.
- Fio
- Tools for development of a workload generation tool for benchmarking and usage of real disk stats.

Preliminary Schedule

Task	Expected date
CODE: Fio workload scripts	Feb. 6, 2017
CODE: Workload VM deployment tool	Feb. 20, 2017
CODE: Workload data analysis scripts	Feb. 27, 2017
WRITE: Synthetic workload section of thesis	Mar. 13, 2017
MILESTONE: Workload generation complete	-----
CODE: Fitness function framework for use with replica placement	Mar. 27, 2017
WRITE: NDFS fitness function implementation section of thesis	Apr. 10, 2017
CODE: Implement weighted random selection algorithms	Apr. 17, 2017
TEST/BENCHMARK: Weighted random pathological case unit test and evaluation	Apr. 24, 2017
WRITE: Weighted random and pathological case section of thesis	May 1, 2017
CODE: Use disk usage and performance stats for placement decisions	May 8, 2017
TEST: Basic unit test simulations to sanity check fitness function	May 15, 2017
MILESTONE: Adaptive data placement implementation complete	-----
TEST: Add instrumentation to code base to troubleshoot odd behavior	May 18, 2017
WRITE: Unit test simulation and instrumentation section of thesis	May 22, 2017

Task	Expected date
BENCHMARK: Generate workloads on multiple clusters and analyze the improvements	Jun. 5, 2017
MILESTONE: Testing and benchmarking on multiple clusters complete	-----
WRITE: Benchmark results section of thesis (include analysis of algorithms)	Jun. 19, 2017
WRITE: Conclusions and future work	Jun. 26, 2017
WRITE: Introduction, Prior work	Jul. 10, 2017
WRITE: Nutanix file system overview	Jul. 17, 2017
WRITE: Finish thesis write-up	Jul. 31, 2017
THESIS APPROVAL	Aug. 28, 2017

Approximate time to completion: 28 weeks

Bibliography

1. Poitras, S. (2015, November 11). The Nutanix Bible - NutanixBible.com. Retrieved February 15, 2016, from http://nutanixbible.com/ (http://nutanixbible.com/)
2. Lakshman, A., & Malik, P. (2008, August 25). Cassandra – A structured storage system on a P2P Network. Retrieved February 15, 2016, from https://www.facebook.com/notes/facebook-engineering/cassandra-a-structured-storage-system-on-a-p2p-network/24413138919/ (https://www.facebook.com/notes/facebook-engineering/cassandra-a-structured-storage-system-on-a-p2p-network/24413138919/)
3. Dean, J., & Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. Communications of the ACM, 51(1), 107-113.
4. Hadoop, A. (2009). Hadoop. 2009-03-06]. http://hadoop (http://hadoop) . apache. org.
5. Xie, J., Yin, S., Ruan, X., Ding, Z., Tian, Y., Majors, J., ... & Qin, X. (2010, April). Improving mapreduce performance through data placement in heterogeneous hadoop clusters. In Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on (pp. 1-9). IEEE.
6. Zaharia, M., Konwinski, A., Joseph, A. D., Katz, R. H., & Stoica, I. (2008, December). Improving MapReduce Performance in Heterogeneous Environments. In OSDI (Vol. 8, No. 4, p. 7).

7. Jin, H., Yang, X., Sun, X. H., & Raicu, I. (2012, June). Adapt: Availability-aware mapreduce data placement for non-dedicated distributed computing. In Distributed Computing Systems (ICDCS), 2012 IEEE 32nd International Conference on (pp. 516-525). IEEE.

8. Perez, J. M., Garcia, F., Carretero, J., Calderon, A., & Sanchez, L. M. (2003, May). Data allocation and load balancing for heterogeneous cluster storage systems. In Cluster Computing and the Grid, 2003. Proceedings. CCGrid 2003. 3rd IEEE/ACM International Symposium on (pp. 718-723). IEEE.

9. Schlierkamp-Voosen, D., & Mühlenbein, H. (1993). Predictive models for the breeder genetic algorithm. Evolutionary Computation, 1(1), 25-49.

10. Baker, J. E. (1987, July). Reducing bias and inefficiency in the selection algorithm. In Proceedings of the second international conference on genetic algorithms (pp. 14-21).

11. Vitter, J. S. (1985). Random sampling with a reservoir. ACM Transactions on Mathematical Software (TOMS), 11(1), 37-57.

12. Efraimidis, P. S., & Spirakis, P. G. (2006). Weighted random sampling with a reservoir. Information Processing Letters, 97(5), 181-185.

13. Chao, M. T. (1982). A general purpose unequal probability sampling plan. Biometrika, 69(3), 653-656.

14. Chaubal, C. (2008). The architecture of vmware esxi. VMware White Paper, 1(7).

15. Borthakur, D. (2008). HDFS architecture guide. HADOOP APACHE PROJECT http://hadoop (http://hadoop) . apache. org/common/docs/current/hdfs design. pdf, 39.

16. Suresh, L., Canini, M., Schmid, S., & Feldmann, A. (2015). C3: Cutting tail latency in cloud data stores via adaptive replica selection. In 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15) (pp. 513-527).

17. Velte, A., & Velte, T. (2009). Microsoft virtualization with Hyper-V. McGraw-Hill, Inc..

18. Lamport, L. (2005). Generalized consensus and Paxos. Technical Report MSR-TR-2005-33, Microsoft Research.s

19. Y. Azar, A. Broder, A. Karlin, and E. Upfal. Balanced allocations. In Proceedings of the 26th ACM Symposium on the Theory of Computing, pages 593{602, 1994.

Glossary

- **Guest VM** - A virtual machine hosted on a hypervisor that is serviced by the CVM.
- **Hypervisor** - Software that runs virtual machines.
- **Oplog** - Persistent write buffer that is part of Stargate.
- **RF** - Replication factor. If the cluster is configured as RF(N), there are N copies of all pieces of data distributed across N nodes.

- **VM** – Virtual Machine