An Application of Hadoop and Horizontal Scaling to Conjunction Assessment

Mike Prausa
The MITRE Corporation
Norman Facas
The MITRE Corporation

ABSTRACT

This paper examines a horizontal scaling approach for Conjunction Assessment (CA) processing using the Apache Hadoop framework. A key focus of this paper is the Input/Output (I/O) characteristics of CA, which directly impact the performance and scalability of the distributed system. To accomplish this and maintain a focus on I/O, a synthetic data model and simulated CA algorithm are used. These components form the basis for a series of scalability tests that result in high collocation rates and near linear scaleout for the 16 node/64 core cluster used. Amdahl's law is used to examine these results as well as provide a basis for extrapolated results. These extrapolated results model larger clusters and identify potential degradation as node count increases.

1. INTRODUCTION

This paper examines a horizontal scaling processing approach for Conjunction Assessment (CA), while considering the impact other algorithms (e.g., Orbit Determination (OD) and ephemeris generation) can have on the overall system from both an Input/Output (I/O) and compute perspective. CA is used to determine potential collisions between Resident Space Objects (RSOs).

In general, scaling falls into two categories: horizontal scaling and vertical scaling. Vertical scaling - also called "scaling up" - approaches scaling from a hardware improvement perspective. This means that the system will satisfy increasing performance requirements through the use of more or higher powered hardware components. In contrast to vertical scaling, horizontal scaling, called "scaling out", approaches scaling with the addition of new, discrete compute nodes. In this case, performance requirements are met by adding an increasing number of compute nodes rather than improving the hardware of a centralized server. An advantage to vertical scaling is its simple development platform while its disadvantages include cost and scaling limits. Conversely, the disadvantage of horizontal scaling is that it is more complex to implement while its advantages are its potentially lower cost and improved scaling capabilities.

In order to facilitate a horizontal scaling system, the Apache Hadoop [1] software stack is used. As a whole, Hadoop is a distributed processing framework that is designed to leverage homogenous or heterogeneously configured commodity hardware for horizontal scaling. Hadoop is particularly well suited for jobs that require the ingest and processing of a large amount of data with the outputs (writes) being small by comparison to the inputs (reads). This "write-once, read-many-times" [2] pattern fits well with CA processing as a large number of RSOs must be read in and processed repeatedly, while the resulting output - the identified conjunctions - is quite small.

The Hadoop Distributed File System (HDFS) is based on the Google File System [2] and is a distributed, disk-based data fabric that allows data to be stored across cluster nodes. By default, HDFS actively groups data into large, sequential blocks of configurable size (a block size of 128MB is used in this study) and type. To satisfy data integrity, HDFS also performs replication of data, which can eliminate the need for a traditional backup system and improve collocation of task and data. The test configuration used in this study uses the three replica HDFS default scheme.

MapReduce [7] is used in the Hadoop software stack as a way to perform distributed processing [6]. A diagram depicting the MapReduce operation for CA is shown below. MapReduce includes two primary phases as shown in Fig. 1. The first is a mapping operation that distributes the tasks (not the data) to the compute nodes. This phase performs the "heavy lifting" of the MapReduce operation and leverages the raw power of the cluster. The second phase is a reduce operation that aggregates results from the mapping operation and performs any necessary post-processing before returning an answer. This operation typically is executed on a much smaller number of cluster nodes than the mapping operation and performs much lighter processing.

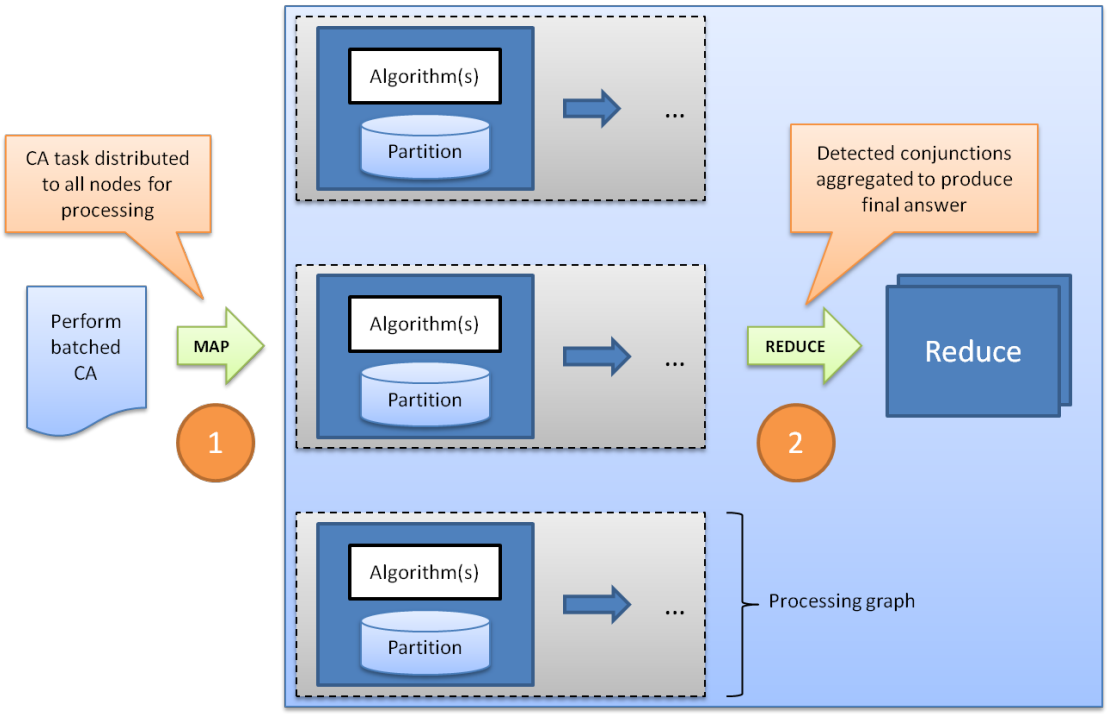

Fig. 1. A visual description of the MapReduce operation for CA

1.1 CA Algorithm

The CA algorithm is used to find conjunctions between *p* primary RSOs and *s* secondary RSOs, which results in *p* * *s* RSO-to-RSO comparisons. For this study, it is assumed that the primary RSOs are a subset of the secondary RSOs, meaning that some comparisons can be removed due to the commutative (comparing RSO 1 vs. 2 is the same as comparing RSO 2 vs. 1) nature of the CA. Additionally, there is no need to compare a RSO against itself. The resulting number of necessary comparisons is given by Eqn. 1 which only holds for a static analysis where the database itself is not changing.

$$\#\ \text{of comparisons} = p * \left(s - \frac{p+1}{2}\right) \tag{1}$$

One way to increase efficiency is by batching the primary RSOs. Without batching, each primary RSO would be compared individually against all secondary RSOs. However, when primaries are grouped into a batch of size *b*, each *batch* is then compared individually against all secondary RSOs. The total number of batches is defined by the ceiling of p/b, *i.e.* $\lceil p/b \rceil$. The use of batching does not decrease the number of comparisons for the static analysis. However, it does reduce the number of times that secondary RSOs are loaded. For example, without batching each secondary must be loaded *p* times, whereas with batching, each secondary must be loaded $\lceil p/b \rceil$ times. This represents a reduction by a factor of ~*b*.

1.2 Amdahl's Law

Scalability was estimated using Amdahl's law [5]. Amdahl's law relates speedup to the number of nodes in the system. In the context of this study, speedup represents the scalability of the system. For a linear scaling system, speedup is proportional to node count. For example, if node count doubles, then speedup should also double. Speedup is given by:

$$S(n) = \frac{t_1}{t_n} \tag{2}$$

where t_1 is the run time for one node and t_n is the run time for *n* nodes. Amdahl's law can then be defined as:

$$S(n) = \frac{n}{1 + \sigma * (n - 1)} \tag{3}$$

where σ represents the seriality, or contention parameter. The seriality parameter is the scalar piece of processing, or the processing that cannot be done in parallel. This parameter represents a bottleneck where one processor must wait for another to finish with the resource before gaining access to it. A common example of this in a modern system is a database lock. If one client has a lock on a piece of shared data, then other clients must wait until this lock is released before they can access it. This behavior results in a serial process and can severely limit the scalability of the system.

The degree to which system speedup is limited can be described by the Amdahl asymptote. This value describes the upper speedup limit of the system. The Amdahl asymptote is equal to the inverse of the seriality parameter, i.e., σ^{-1} [4].

Fig. 2 depicts the components of Amdahl's law graphically. The green line represents an ideal system that does not contain any bottlenecks. In other words, the seriality parameter is zero (σ = 0). The dashed, red line is equal to the inverse of the seriality parameter (Amdahl asymptote) and describes the upper limit of system speedup. The blue line represents actual system speedup using Amdahl's law given byEqn. 3.

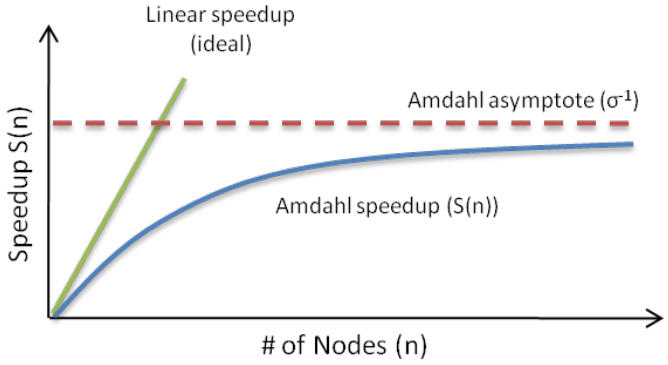

Fig. 2. Speedup vs. node count in context of Amdahl's law

Another metric that helps to characterize system performance and scalability is efficiency. This metric identifies the degree to which the system speedup is adhering to linearity. Efficiency is given by:

$$E(n) = \frac{S(n)}{n} \tag{4}$$

An alternate quantity to examine is the scaleup of the system, which should not be confused with the vertical scaling term “scaling up” discussed previously in Section 1. In the case of Amdahl’s law, scaleup is dictated by the throughput given by:

$$X(n) = \frac{c_n}{t_n} \tag{5}$$

where c_n is the number of completed transactions for the *n* node case. Scaleup is then given by:

$$C(n) = \frac{X_n}{X_1} \tag{6}$$

This equation is intended to normalize the throughput values. As node count increases, the scaleup value should also increase linearly.

In this study, an arbitrary node count is used as the starting point rather than a node count of one. To accommodate starting at any node count, n* is defined by:

$$n^* = \frac{n}{n_0} \tag{7}$$

where n_0 is the minimum number of nodes. In this case, n* replaces *n* in all previous equations. Additionally, *n* can be replaced by any arbitrary parameter.

1.3 Node Count and Throughput

This study uses node count instead of processor core count to relate system scalability. Nodes imply a particular hardware configuration including both compute resources such as the number of CPU cores available and I/O resources such as the number of HDDs available. With a framework like Hadoop, this discernment becomes particularly important as it details the configuration of the system. For example, a system could have 96 cores in six nodes and perform poorly due to I/O constraints, such as a limited number of HDDs. By comparison, a system could have 96 cores in 24 nodes and have much better performance and lower I/O constraints due to several more HDDs. In short, the node configuration - versus just processor core counts - becomes an important element to understand when interpreting the metrics.

Throughput is a second metric that is important to understand when interpreting the metrics presented in this study. At first glance, a throughput metric describing the number of RSOs processed per minute (RSOs/minute) for CA sounds reasonable. However, this particular metric is linked to catalog size and can make throughput comparisons for differing catalog sizes difficult. For example, a throughput of 200 RSOs/minute for a 1,000 satellite catalog (SATCAT) represents much less work than the same throughput rate for a 100,000 SATCAT.

Using the above example, a throughput rate of 200 RSOs/minute for a 1,000 SATCAT represents 179,900 comparisons/minute. In contrast, the same 200 RSOs/minute throughput rate for a 100,000 SATCAT is equal to 19,979,900 comparisons/minute, or approximately two orders of magnitude more comparisons/minute than the 1,000 SATCAT. It becomes readily apparent that a throughput rate of RSOs per minute is not a good metric to use when comparing differing SATCAT sizes. Instead, the measure of comparisons per minute is a much more accurate metric as it is independent of catalog size.

2. STUDY DESIGN AND METHODOLOGY

The data model used in this study models the size and relationships of RSO and ephemeris data without the complexity of the "real" data. In order to efficiently store object data in HDFS, the Hadoop Writable interface was implemented to serialize only data that needed to be persisted rather than serializing the entire object. The implementation of this interface is the recommended approach for storing object data in HDFS [3] as it decreases the amount of data that is persisted to HDFS. These smaller data sets require less overall storage and thus, less I/O, which improves performance. However, this is accomplished at the expense of development time as well as code independence. Methods adhering to the Hadoop Writable interface that read and write only specific data must be implemented manually by the developer.

A RSO's ephemeris is the primary piece of data used in the model. Each RSO in the 100,000 SATCAT contains a single ephemeris object consisting of 10,000 individual ephemeris points. Each ephemeris point contains simulated data for: time, position, and velocity. Covariance data is not stored as it is assumed this data will be generated on the fly once a potential conjunction has been detected. The physical size of the ephemeris for each RSO is 1.08MB, which translates into an entire 100,000 SATCAT size of approximately 105.4GB. Considering HDFS maintains three data replicas by default, the total physical footprint of the catalog used in this study is approximately 316GB.

The computation normally performed by the operational CA algorithm was simulated. Each primary and secondary RSO was processed iteratively, similar to how traditional CA algorithms operate, but the actual computations that occur during a comparison were not performed. Instead, the processing associated with a comparison was mimicked by executing a computation cycle to evoke a high CPU utilization rate. This action introduced more realism to the test as the high CPU utilization rate can cause a decrease in I/O contention, which impacts system throughput.

The use of a simulated CA compute time also allowed for a further characterization of the system, as the system can shift between being I/O bound to being CPU bound. A more accurate cluster configuration can be determined depending on which side of the spectrum the system falls. If more I/O bound, the nodes may benefit from more or faster mechanical HDDs or Solid State Drives (SSDs). If more CPU bound, compute nodes with more cores or CPUs may be more beneficial to increase compute capacity

2.1 Organization of Data

A well designed data partitioning strategy is critical to successfully implementing and operating a horizontally scaling system. If data is not partitioned correctly on the cluster, bottlenecks can quickly occur where one node may host data that several other nodes require for processing. It is imperative to place data across the cluster in a way to minimize this type of contention and maximize parallelization.

This study uses a data placement strategy that is RSO-centric as shown below in Fig. 3. The term "RSO-centric" means that data is grouped together by RSO number. For example, ephemeris data for a particular RSO is composed of 10,000 discrete ephemeris points that are used to represent a RSO's ephemeris. As the data placement strategy used is RSO-centric, all of the 10,000 ephemeris points for a particular RSO's ephemeris are grouped together and associated with a specific RSO number. In the end, each node in the cluster contains a small collection of unique RSO-linked ephemeris data, which represent a piece of the overall SATCAT. The entire SATCAT can be realized when accounting for each node's partitioned catalog of RSOs. When a CA is performed on a node, it processes *all* ephemeris for a *subset* of the RSOs in the SATCAT.

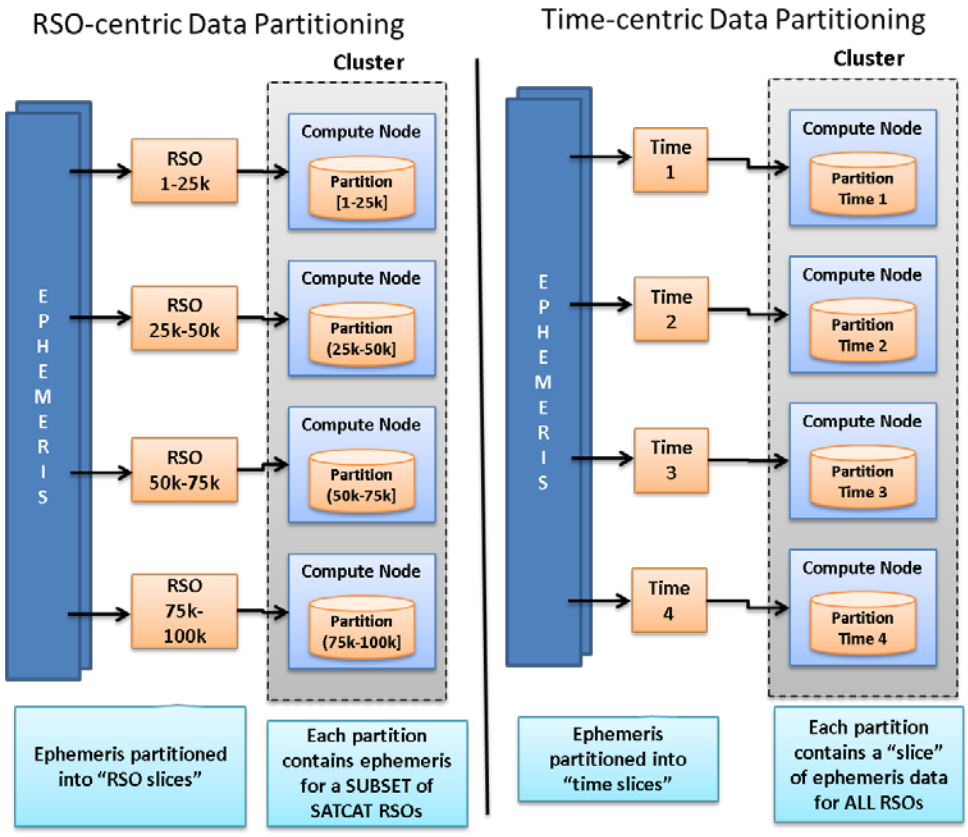

Fig. 3. Data partitioning (left) RSO (right) Time-centric

An alternative approach to grouping ephemeris data together by RSO is to group this data together by a segment of time, or leverage a time-centric approach (Fig. 3). To use a simple example, consider that ephemeris is typically generated for a five day period. Assuming an even distribution over the 10,000 total ephemeris points, each day would then be represented by 2,000 points of ephemeris data. Each of these 2,000 point time "slices" could then be

distributed to separate nodes. In the case of a five node cluster, each node would receive 2,000 ephemeris points plus a small number of "overlap" points required for algorithmic processing. If the number of available nodes is 10, then each node would receive 1,000 points (plus overlap points) and so on. When a CA is performed on a node, it processes *a subset* of the ephemeris for *all* RSOs in the SATCAT.

Advantages and disadvantages exist for both RSO-centric and time-centric data placement strategies. For example, a RSO-centric approach is not as advantageous to a distributed CA algorithm as is a time-centric approach. In the case of a RSO-centric strategy, the local node may or may not contain data for the primary RSOs. If it does not contain the correct data, it must either retrieve the data from a remote node, or it must retrieve the RSO's state vector and compute the ephemeris locally. Retrieving data from a remote location can increase network contention and latency leading to performance and scalability degradation. Computing the ephemeris locally can significantly impact local system performance, which in turn impacts overall system performance.

By comparison, a time-centric approach eliminates the need to remotely populate or compute the primary RSOs' ephemeris as all of the data needed for the primaries and secondaries is available on the local node. As data is partitioned by time, CAs are performed over only a portion of a RSO's ephemeris data defined by the time partitions, so each node contains all of the data it needs to operate autonomously from other nodes. In short, no data exchange is needed, which makes this scenario ideal for supporting an embarrassingly parallel processing system.

However, a time-centric approach may be suboptimal when considering other algorithms such as OD and ephemeris generation. The OD process updates a RSO's state vector and then starts a subsequent algorithm to generate ephemeris for that RSO. When considering a RSO-centric approach, this isn't an issue; the ephemeris is generated and a single write takes care of persisting it to the cluster. However, the issue becomes more complex when considering a time-centric approach. The ephemeris for a RSO would need to be broken into a series of time "slices". Depending on the partitioning strategy, each of these time slices would necessitate a separate write operation to persist to the desired cluster node. If time was partitioned into five segments, then five separate writes would need to occur. However, if time was partitioned into 50 segments, as with a 50 node cluster, 50 separate writes may need to occur. While a time-centric data partitioning strategy is certainly an attractive alternative for CA, there are some potential challenges that must be addressed before it can be used efficiently with OD and ephemeris generation.

A RSO-centric data partitioning strategy presents challenges for populating the primary RSO ephemeris data. This can be addressed with one of two approaches: 1.) retrieving the remote ephemeris data for each primary RSO or 2.) retrieving the remote state vector for each primary RSO and computing the ephemeris data locally. The first approach impacts performance via an increase in I/O and the second approach impacts performance via an increase in I/O (to a lesser degree) and an increase in CPU usage. An alternate approach is to use a time-centric data partitioning strategy that eliminates this problem entirely, but can potentially impact the performance of other algorithms, such as OD. Due to the potential complexities of implementing either of these solutions, as well as hardware limitations (particularly limited RAM), this study does not populate the primary RSO ephemeris data as discussed above. Instead, each RSO object references a static ephemeris object that still contains 10,000 ephemeris points, but is common to all of the primary RSOs. This is an important limitation to note as it can directly impact system coherency and affect scalability.

2.2 Equipment

The hardware used in this study consists entirely of a Penguin Computing High Performance Computing (HPC) cluster. This cluster is configured to include a maximum of 16 compute nodes and one head node. Each of the 16 compute nodes includes the following hardware: two Dual core AMD Opteron 2214 CPUs running at 2.2GHz each, 4 GB DDR2 Random Access Memory (RAM), 1 Gbps Ethernet, and one 7200 RPM SATA I hard disk drive. The single head node used for this study is a Penguin Computing Altus 2600SA with the following configuration: two Dual core AMD Opteron 2214 CPUs running at 2.2GHz each, 4 GB DDR2 RAM, 1 Gbps Ethernet, and three 15K RPM SAS drives in a RAID 5 configuration. The following software is used on the compute nodes and the head node: 64-bit CentOS version 5.5, Java 64-bit server VM 1.6.0_20-b02, and Hadoop 0.20.2.

3. STUDY RESULTS AND ANALYSIS

Three sets of tests were performed. The first set examined system scalability in order to understand the degree to which the system scales as additional compute nodes are added. Ideally, system scalability will be linear, meaning that speedup will double when node count doubles. However, this is not expected due to overhead and seriality (σ) that is present in any practical system. System scalability was examined by varying the node count.

The second series of tests explored the effect of increasing catalog size, which is synonymous with increasing the amount of data managed by the system. The purpose of this test is to identify processing inconsistencies as overall data size grows. Assuming the system is processing data in a balanced manner, the throughput (comparisons/minute) should be similar for all test cases.

The final set of tests examines the CA comparison compute time from both an I/O and CPU perspective. As the comparison compute time of the operational algorithm is unknown for the hardware profile used in this study, a range of possible values is used. The results of these tests play a significant role in characterizing the I/O and CPU bound nature of the system.

3.1 Scalability Tests

Scalability for node counts of two, four, eight, and 16 nodes were examined. These results are shown in Table 1. As expected, the total run time decreased as more nodes were added. The average measured compute time was always above the desired simulation compute time, which may result in a slight, artificial decrease to system performance.

Table 1. Results for variable node count

# of Nodes (4 cores/node)	Total Run Time (min)	Average Compute Time(ms)	Collocated (%)	Speedup S(n*)	Efficiency E(n*)
2	65.71	0.0589	100.00%	1.00	1.00
4	33.04	0.0590	99.88%	1.99	0.99
8	16.80	0.0592	99.76%	3.91	0.98
16	8.65	0.0592	98.84%	7.60	0.95

Speedup (S(n*)) and efficiency (E(n*)) were computed from the data in Table 1 using $n_0 = 2$ nodes. Amdahl's law (Eqn. 3) was applied to this data and σ was found to be 0.0076. From this, Amdahl's asymptote was calculated to be 132. Efficiency was found to decrease as node count increased, which corresponds to the decrease in collocation, but also suggests increasing overhead. In the experimental region of two to 16 nodes, the data appears to exhibit near linear scalability (Fig. 4a and Fig. 4c). However, when extrapolating to much larger node counts, the non-linearity becomes evident (Fig. 4b and Fig. 4d).

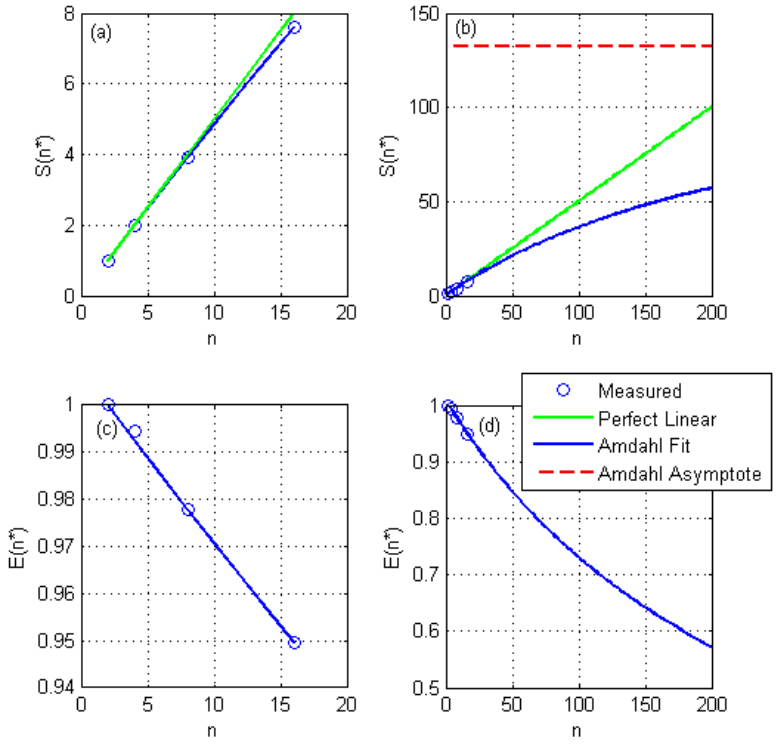

Fig. 4. Speedup (S(n*)) and Efficiency (E(n*)) of the system

Collocation was found to decrease as node count increased (Table 1); however, the data is still highly collocated at 98.84% for the 16 node test case. Regardless, this decreasing collocation rate is one of the prime contributors to the non-linear scalability observations noted previously. This is likely due to the decrease in replica-to-node ratio. As data is replicated three times for all four tests, processing conducted in the two node test will always have local data (100% collocation). However, for the 16 node test, this is not the case as only three of the 16 nodes will contain a replica, resulting in a higher chance of a non-data local task execution. These non-data local task executions directly impact system scalability as they are significantly slower than a data-local (collocated) task execution. Due to MapReduce's desire to execute task and data locally, these non-collocated executions occur primarily during the end of the test run. During this transient phase, nodes may begin processing remote data in an effort to complete the overall MapReduce job faster. This is typically beneficial versus the alternative of waiting for the remote nodes hosting the data to become available and process the data locally.

3.2 Catalog Scalability

Scalability with change in catalog size was examined on the 16 node cluster for catalog sizes of 12.5k, 25k, 50k, and 100k RSOs. These results are shown in Table 2. As expected, the total run time increased with the catalog size. The throughput and scaleup metrics were calculated based on the modified catalog size (*cat**) where cat_0 = 12,500 RSOs. Both throughput (X(*cat**)) and scaleup (C(*cat**)) were found to increase with increasing catalog size. This may be due to the fixed time costs of the system being relatively less important for longer run times. These fixed time costs will factor in more significantly as the overall run time decreases. This behavior becomes less noticeable once a catalog size of 50,000 RSOs is reached. A settling of the scaleup value is desirable as it confirms that cluster processing is relatively balanced as catalog and overall data size increases.

Table 2. Experimental test results for increasing catalog size

Catalog Size (# of RSOs)	Total Run Time (min)	Average Compute Time (ms)	Number of Comparisons	Throughput X(cat*)	Scaleup C(cat*)
12,500	1.23	0.0598	39,841,875	32,512,036	1.00
25,000	2.29	0.0591	86,716,875	37,908,740	1.17

50,000	4.24	0.0593	180,466,875	42,521,323	1.31
100,000	8.53	0.0591	367,966,875	43,120,276	1.33

3.3 CA Comparison Compute Time

A critical parameter that is unknown in this study is the CA comparison compute time. This is the amount of time required by the CPU to compare two RSOs' ephemeris and determine if a conjunction is present or not. The CA comparison compute time becomes an important component as it drives the system to be I/O bound or CPU bound depending on its value.

As the actual algorithmic computation time of this parameter is unknown, a series of tests were run to cover a range of potential values from 0ms to 0.5ms per CA comparison. These results are shown in Table 3 where the Comparison Compute Time is the specified comparison compute time, Average Measured Compute Time is the actual measured comparison compute time, and Total Run Time is the time, in minutes, required for the entire test run including both I/O and compute time. The average measured compute time was found to be 0.01ms more than the specified compute time. This is likely due to latencies within the local machine and results in the total run time being slightly greater than if the comparison time were exact. In general, the 0.01ms excess compute time results in values that are more conservative than if the compute time were exact.

Table 3. Compute time study results

Comparison Compute Time (ms)	Average Measured Compute Time (ms)	Total Run Time (min)
0	0.00	5.26
0.01	0.01	5.35
0.02	0.03	5.90
0.03	0.04	6.64
0.05	0.06	8.53
0.1	0.11	13.49
0.3	0.31	33.09
0.5	0.51	52.95

Fig. 5 shows the total run time versus the average measured compute time. For large compute times (red points), the response is linear, while for small compute times (blue points), the response is non-linear (not to be confused with linear/non-linear scalability). This change in response is likely due to I/O and CPU bottlenecks on a node.

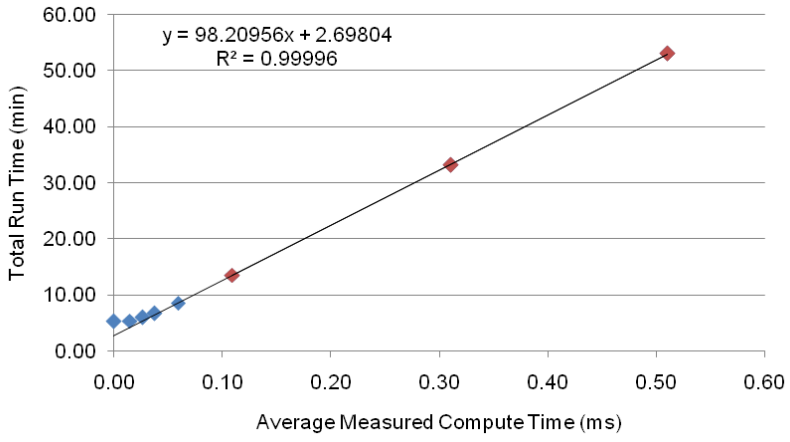

Fig. 5. Comparison compute time vs. total run time

For small compute times, the bottleneck is based on the I/O rate resulting in an I/O bound system. For large compute times, the bottleneck is based on CPU speed, resulting in a CPU bound system. In between the two regions, the system is bound by both I/O and CPU. Fig. 6 shows the possible split based on this data set.

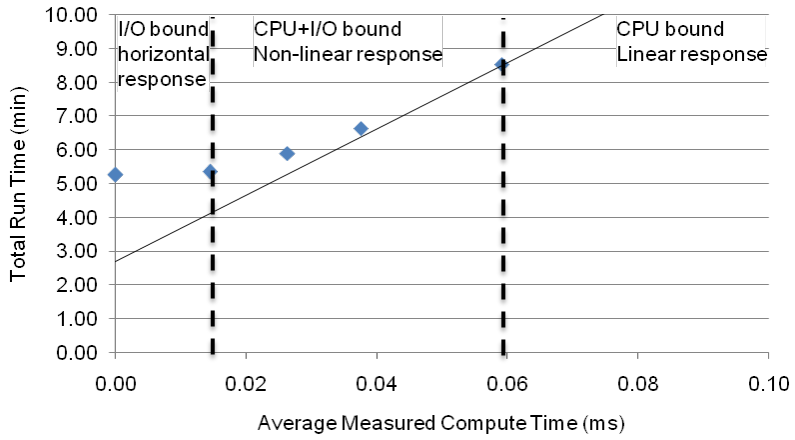

Fig. 6. I/O and CPU bound regions based on varying comparison compute times

The data presented in Fig. 6 becomes particularly important when sizing and configuring a cluster to address system requirements. If the actual algorithmic compute time falls in the I/O bound region of Fig. 6, then the compute nodes would benefit from having more I/O capacity. If the actual compute time falls on the opposite side of the graph, in the CPU bound region, the compute nodes would benefit from greater CPU capacity. If the actual compute time falls in the middle region, a more balanced I/O and CPU compute node configuration would be optimal.

4. CONCLUSION

This study explored the I/O and scalability aspects of CA using a synthetic data model and synthetic processing application. The results obtained within this study describe the scalability characteristics of CA for a 100,000 SATCAT using the Hadoop horizontal scaling framework.

The node scaling experimental results indicated near-linear scaling for the 16-node/64-core cluster used in this study. However, extrapolation of these results shows an increasing deviation from linearity, particularly as node count approaches 200. Experimental results for catalog scaling show system throughput stabilization once a 50,000 SATCAT is reached. This indicates that processing has reached a steady state and is occurring in a relatively balanced manner. The final test highlights the impact that the CA comparison compute time can have on a system. If the comparison compute time is low, the system becomes more I/O bound. If the time is high, the system becomes more CPU bound. This value becomes important when specifying hardware components for optimal processing.

5. REFERENCES

1. The Apache Software Foundation., Hadoop. *hadoop.apache.org.* [Online] May 11, 2010. [Cited: September 24, 2010.] http://hadoop.apache.org/.
2. Ghemawat, Sanjay, Gobioff, Howard and Leung, Shun-Tak., "The Google File System." ACM SIGOPS Operating Systems Review, New York, NY : ACM, 2003, Issue 5, Vol. 37. 0163-5980.
3. White, Tom., *Hadoop: The Definitive Guide.* [ed.] Mike Loukides. First Edition. Sebastopol, CA : O'Reilly Media, Inc., 2009. 978-0-596-52197-4.
4. Gunther, Neil J., *Guerilla Capacity Planning.* s.l. : Springer-Verlag, 2007.
5. Amdahl, Gene., "Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities." 1967. Proc. AFIPS Conf. Vol. 30, pp. 483-485.
6. White, Tom., "MapReduce." *weblogs.java.net.* [Online] September 25, 2005. [Cited: September 28, 2010.] http://weblogs.java.net/blog/tomwhite/archive/2005/09/mapreduce.html.
7. Dean, Jeffrey and Ghemawat, Sanjay., "MapReduce: Simplified Data Processing on Large Clusters." San Francisco, CA : s.n., December, 2004. OSDI'04: Sixth Symposium on Operating System Design and Implementation.