HAMR Technical Report: Graph

K-Cliques

HAMR vs
Hadoop and Spark
Version 0.3
Nov 10, 2014

hamr-info@hamrtech.com
www.hamrtech.com
302.797.4267

Key Findings: ***HAMR™ outperforms both Hadoop and Spark by up to 14x.***

HAMR manages explosive intermediate data with significantly better efficiency than both Hadoop and Spark. HAMR is fundamentally a streaming engine, allowing intermediate data to be pushed through the execution graph without piling up. These efficiencies translate into speedups of up to 14x versus Hadoop and 3x versus Spark. HAMR can handle scaling data size well beyond the capabilities of Spark.

HAMR is able to achieve these improvements with a simple API that does not require the user to write a more complex program.

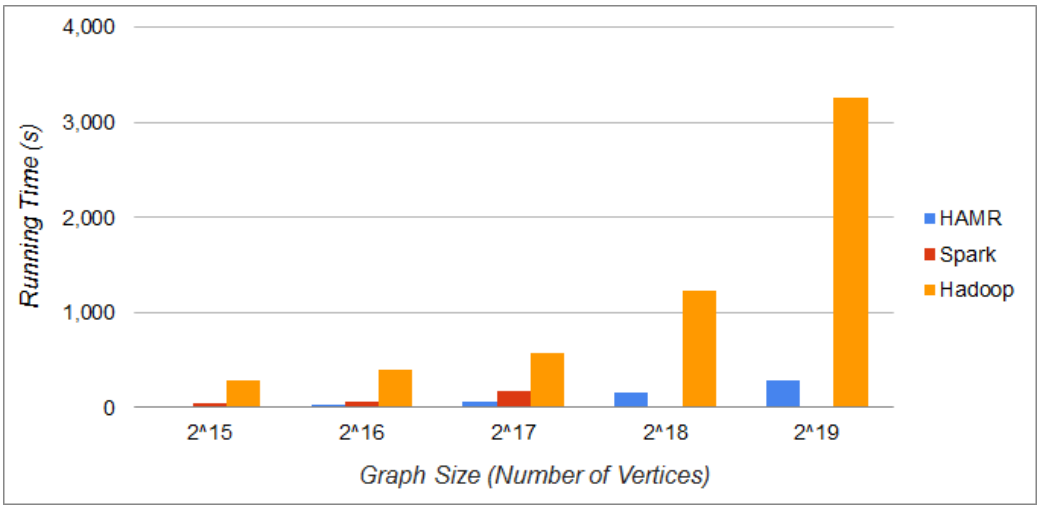
Introduction

A K-clique is defined as a fully connected set of K vertices in a graph. As a social network example, if Alice, Bob, and Claire are all friends with each other, these three people are in a clique of size 3. A K-clique query searches a given graph to identify all cliques of size K. In a social context, given a social network graph where the vertices are users and the edges are friendship relations, one line of output is generated for each set of K people that are all friends with each other. K-Cliques finds application in a wide range of fields including social engineering, networking, and routing. This document is a comparison between Apache Hadoop, Apache Spark and HAMR running the K-cliques benchmarks developed by HAMR Analytic Technologies.

Results

The following graph presents the performance comparison between HAMR, Hadoop, and Spark running the K-Cliques implementations.

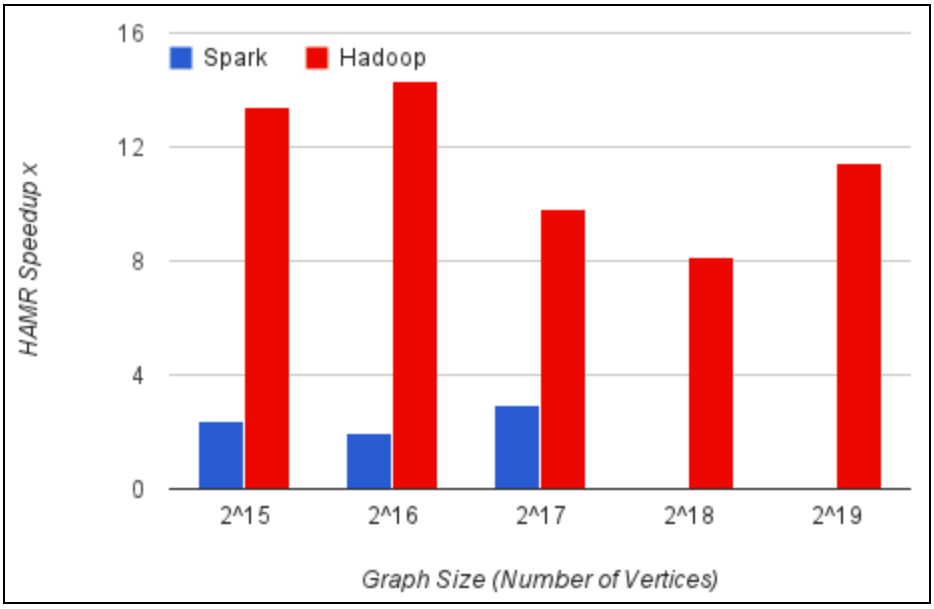
K-Cliques (K=3): HAMR, Spark, and Hadoop



For graphs between 32K and 524K vertices, HAMR clearly outperforms Spark and Hadoop. At 128K vertices Spark begins to suffer from garbage collector pauses, causing the running time to disproportionately

increase. At 256K vertices Spark was unable to reliably complete the execution, frequently throwing OutOfMemoryError exceptions, and finally failing after half an hour of retries.

The following diagram and table summarize HAMR's speedup[1] over Hadoop and Spark. HAMR is 14x faster than Hadoop on the 64K vertex graph.



	Speedup over Hadoop	Speedup over Spark
Minimum	8.14	1.99
Maximum	14.30	2.95
Average	11.42	2.44

Improvement Sources

K-Cliques produces an explosive amount of intermediate data, generating candidate cliques that must be verified and filtered. Hadoop must save this intermediate data to disk, while Spark must store the intermediate data in an RDD before it can be reduced. HAMR is able to stream the intermediate data through the workflow while it is being generated, allowing HAMR to minimize the amount of intermediate data that must be stored in memory.

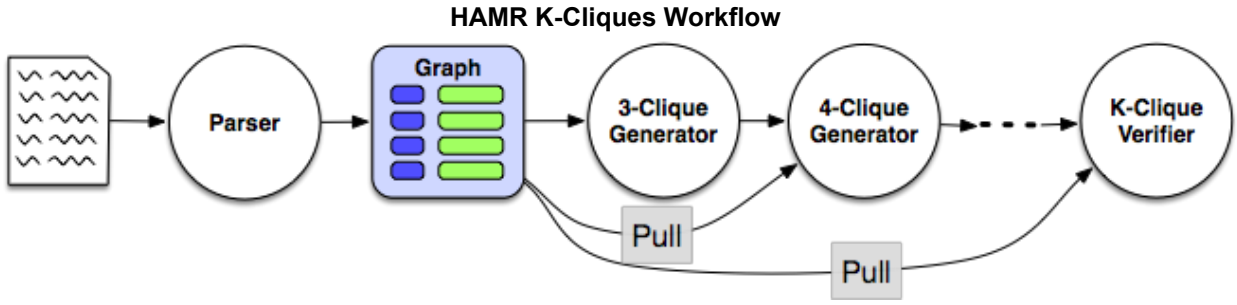
HAMR Support for K-Cliques

HAMR's implementation of K-Cliques demonstrates the ability to stream data through a workflow, the ability to push and pull key/value pairs, and the ability to join key/value pairs with reference data. First the edges are read from a file as lines of text. These lines of text are parsed into vertex/vertex pairs which are then

[1] (Hadoop | Spark) / (HAMR running time)

used to build an edge list for each vertex. The combination of all edge lists is a graph, which is stored in a Key/Value Store (KVS), a partitioned and distributed data structure. Each of the edge lists is assigned to a particular partition and stored within the local memory of the node that owns that partition.

The remainder of the workflow waits for the graph KVS to be completely populated. The ability to delay execution until data dependencies are satisfied is a property of a KVS. Next each edge list is pushed to a Transform that generates candidate 3-cliques. The generated key/value pairs, each representing a candidate 3-clique, are sent to the next Transform where it is determined if this candidate is an actual 3-clique (do all vertices have edges to each other). This determination requires the edge list for the incoming key (vertex) to be pulled from the graph. If the candidate is an actual 3-clique then candidate 4-cliques are generated and the process continues. The final stage determines whether the candidate K-clique is in fact a K-clique. If the key/value pair is a K-clique then that clique is output to a file. This workflow diagram for this process is shown in the next figure.



HAMR's dynamic task execution engine allows higher priority tasks to take precedence. This engine is optimized to push data through the workflow so that it can be reduced to a final result which is written to disk, allowing a significant reduction in the amount of outstanding intermediate data. Work is aggregated between Flowlets, and Flowlets further down the workflow graph are given higher priority than those further up the graph. All this is done without the need for tuning or intervention by the developer, creating a very efficient and easy to use system for processing big data.

Conclusion

HAMR is significantly more efficient than Hadoop and Spark in both memory and processor utilization. This efficiency translates into performance speedups of up to 14x versus Hadoop and 3x versus Spark. HAMR is capable of executing data sets up to 10x larger than Spark in memory, and can execute even larger datasets without modification to the source code. HAMR is able to achieve these improvements with a simple API that does not require the user to write a more complex program. The HAMR implementation of K-Cliques is approximately the same number of lines of source code as the Spark implementation. This demonstrates that the ease of application development is roughly comparable between the two systems.

Testbed Description

We compare our HAMR implementation of K-Cliques to an equivalent Hadoop and Spark implementation. All three implementations were developed by HAMRTech.

We report execution time in seconds collected as an average of three runs for graphs ranging from 32 thousand to 512 thousand vertices.

HAMR, Hadoop, and Spark read the input graph from the HDFS file system and write results back to HDFS. We configure Hadoop and HDFS according to the convention set by Hortonworks (http://goo.gl/4XcCbY). The HDFS block replication is set to 3 with a block size of 128 MB. While these parameters are tweakable for a given benchmark, we attempt to use a configuration most representative of standard clusters.

Spark was executed using YARN with 4 executors, 4 GB of driver memory, 16 GB of executor memory, and 16 executor cores. This configuration provided optimal performance across all runs.

The testing environment is a four node cluster with the specifications listed below. The benchmark was executed on the same cluster, with the same number of compute nodes, and with the same input data for Hadoop, Spark, Spark with GraphX, and HAMR.

- System: 4 Node Dell PowerEdge R420 2 x Xeon® E5-2450 32 GB 1600 MHz DDR3, 4 x SATA 2 (3 GB/s)
- Network: 4 x QDR Infiniband
- OS: CentOS release 6.5
- Java: JDK 1.7.0 u55

HAMR™ is the next generation in-memory real-time software appliance enabling out of the box Big Data analytics. While maintaining the best aspects of MapReduce found in Apache Hadoop 2.0, HAMRTech has developed a novel Big Data programming model and runtime inspired by MIT dataflow. HAMR leverages innovative Flowlets™ and Key/Value Stores to reduce idle time and costly spilling of data to disk, saving both time and energy. Furthermore, HAMR provides access to all enterprise data, and supports batch, streaming, and real-time analytics, making HAMR a fast, efficient, and more complete Big Data solution.