Nobody ever got fired for using Hadoop on a cluster

Antony Rowstron Dushyanth Narayanan
Austin Donnelly Greg O'Shea Andrew Douglas
Microsoft Research, Cambridge

Abstract

The norm for data analytics is now to run them on commodity clusters with MapReduce-like abstractions. One only needs to read the popular blogs to see the evidence of this. We believe that we could now say that "nobody ever got fired for using Hadoop on a cluster"!

We completely agree that Hadoop on a cluster is the right solution for jobs where the input data is multi-terabyte or larger. However, in this position paper we ask if this is the right path for general purpose data analytics? Evidence suggests that many MapReduce-like jobs process relatively small input data sets (less than 14 GB). Memory has reached a GB/$ ratio such that it is now technically and financially feasible to have servers with 100s GB of DRAM. We therefore ask, should we be scaling by using single machines with very large memories rather than clusters? We conjecture that, in terms of hardware and programmer time, this may be a better option for the majority of data processing jobs.

Categories and Subject Descriptors C.2.4 [*Distributed Systems*]: Distributed applications

General Terms Performance

Keywords MapReduce, Hadoop, scalability, analytics, big data

1. Introduction

We are all familiar with the saying "nobody ever got fired for buying IBM". In the data analytics world, are we at the point where "nobody ever got fired for using Hadoop on cluster"? We understand that there are many jobs that process exabytes, petabytes, or multi-terabytes of data that need infrastructures like MapReduce [1, 6] (Hadoop) running over large clusters of commodity servers. However, evidence suggests that the *majority* of analytics jobs do not process huge data sets. For example, as we will discuss in more detail later, at least two analytics production clusters (at Microsoft and Yahoo) have median job input sizes under 14 GB, and 90% of jobs on a Facebook cluster have input sizes under 100 GB.

We also speculate that many algorithms are complex to scale out and therefore expensive in terms of human engineering. Many important analytics jobs, for example iterative-machine learning algorithms, do not map trivially to MapReduce. Typically the algorithm is mapped into a series of MapReduce "rounds", and to achieve scalability it often has to be approximated thus sacrificing accuracy and/or increasing the number of MapReduce rounds.

Finally, we observe that DRAM is at an inflection point. The latest 16 GB DIMMs cost around $220, meaning 192 GB can be put on a server for less than half the price of the server. This has significant cost implications: increasing the server memory allowed 33% fewer servers to be provisioned, it would reduce capital expenditure as well as operating expenditure. Further, Moore's Law benefits memory as well, so every eighteen months the price drops by a factor of two. Next year we would expect to be able to buy 384 GB for the same price as 192 GB today.

In this position paper we take the stand that we should not automatically scale up all data analytics workloads by using clusters running Hadoop. In many cases it may be easier and cheaper to scale up using a single server and adding more memory. We will provide evidence of this using some of our own experiences, and we also note that has been recognized for a long time by the in-memory database community. However, in the MapReduce/Hadoop data analytics community this message seems to have been drowned out.

To support this we consider *(i)* memory costs, *(ii)* job input sizes, *(iii)* implementation complexity and *(iv)* performance. The first two are based on empirical data. For the complexity, we examine AdPredictor, a machine learning algorithm that processes click logs to generate click-through predictions for a search engine. AdPredictor makes tradeoffs between execution time and accuracy in order to achieve scalability. We compare the performance of AdPredictor on both a single large-memory server and a cluster, and also compare implementations using different programming abstractions. We also compare the single-server and cluster approaches for an embarrassingly parallel synthetic benchmark that is a "best case" for MapReduce.

HotCDP 2012 - 1st International Workshop on Hot Topics in Cloud Data Processing
April 10, 2012, Bern, Switzerland.

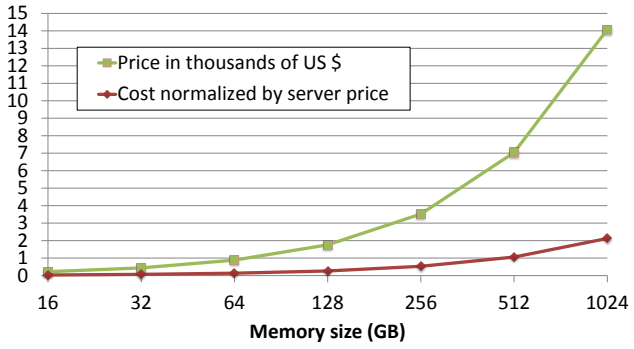

Figure 1. Absolute and relative price of DRAM

2. Memory costs and job input sizes

Memory price/performance has a reached a tipping point. The price of memory has dropped considerably just in the last one year, as it benefits from Moore's Law plus the increased consumer demand from servers for larger memory DIMM form factors. Currently, a 16 GB ECC DIMM can cost as little as US$220[1]. Note that this is a retail price for an average person on the street from a large commodity vendor, and does not include bulk or preferred-vendor discounts. Based on this memory module, Figure 1 shows how much it costs to provision different memory sizes. A mere 192GB, commonly supported on motherboards supporting 12 or more DIMMs, costs approximately $2500. Figure 1 also puts this into further perspective by showing the price normalized by the price of a commodity rack-based server advertised at $6638[2]. Upgrading this server to 192 GB would cost $2640, i.e. 40% of the server price. Hence if this upgrade improves server performance by more than 40% we will have reduced overall capital expenditure. As we will show later, a single "big memory" (192 GB) server we are using has the performance capability of approximately 14 standard (12 GB) servers. In other words, we could replace 14 servers by one big-memory server with 192 GB for about an eighth of the total cost.

We analyzed 174,000 jobs submitted to a production analytics cluster in Microsoft in a single month in 2011 and found that the median job input data set size was less than 14GB. Elmeleegy [7] analyzes the Hadoop jobs run on the production clusters at Yahoo. Unfortunately, the median input data set size is not given but, from the information in the paper we can estimate that the median job input size is less than 12.5 GB[3]. Ananthanarayanan et al. [3] show that Facebook jobs follow a power-law distribution with small jobs dominating; from their graphs it appears that at least 90% of the jobs have input sizes under 100 GB. We therefore believe that there are many jobs run on these clusters which are smaller than the memory of a single server.

3. AdPredictor and platforms

AdPredictor is a machine learning algorithm that predicts the click-through rate (CTR) for sponsored search on Bing. It is based on a probit regression model that maps discrete or real-valued input features to probabilities [8]. It is naturally expressible as a factor graph over which inference is done using expectation propagation (EP). The size of the factor graph scales with the size of the input click log, since each log entry introduces at least one edge into the graph. Typically multiple rounds of EP are run on a given set of inputs until the model probabilities converge. ADF (assumed density filtering) algorithm [8] is an optimization that makes the model state independent of the click log size, and allows the click log data to be streamed once to build the model. However the tradeoff is that ADF is an approximation of EP. Coming up with this approximation is non-trivial and judged worthy of a paper at a machine learning conference [8]. Thus ADF adds complexity and impacts accuracy to gain performance.

Scaling out the EP and the ADF algorithm to a MapReduce cluster introduces further complexity and accuracy loss. We have mapped both algorithms to a MapReduce like abstraction. The input click log is sharded and a local model is computed on each partition by iterating over the shard data. The local models are then merged to create a global model. A single MapReduce round does not give an accurate model, since it does not capture all the interactions between data in different shards. Hence multiple MapReduce rounds must be run, with the result of each round being used as the "prior" by the mappers in the next round. A strict MapReduce model forces mappers and reducers to be stateless. State that persists across MapReduce rounds must be written to shared storage and read back. This is particularly expensive for EP where the state size scales with the input data. To reduce this cost, we have implemented an extended "MapReduce++" platform that allows state to be retained in memory across rounds. Using MapReduce requires non-trivial additional programming effort and is a less natural fit for these algorithms than the factor graph. To explore scaling out without sacrificing the factor graph representation, we have also implemented EP on CamGraph, a distributed infrastructure designed specifically to support graph algorithms. CamGraph automatically partitions the factor graph using well known techniques such as *equals factors* and allows message passing within and across partitions.

[1] Kingston 16GB 240-Pin DDR3 SDRAM ECC Registered DDR3 1066 Server Memory QR x4 w/TS Model KVR1066D3Q4R7S/16G from `http://www.newegg.com/` accessed on 6th January 2012

[2] HP ProLiant DL360 G7 X5650 1U Rack Server with 2 Xeon X5650s (2.66 GHz - 6 cores), 12 GB of RAM (max 384 GB) from `http://www.newegg.com/` accessed on 6th January 2012.

[3] The paper states that input block sizes are usually 64 or 128 MB with one map task per block, that over 80% of the jobs finish in 10 minutes or less, and that 70% of *these* jobs very clearly use 100 or fewer mappers (Figure 2). Therefore conservatively assuming 128 MB per block, 56% of the jobs have an input data set size of under 12.5 GB.

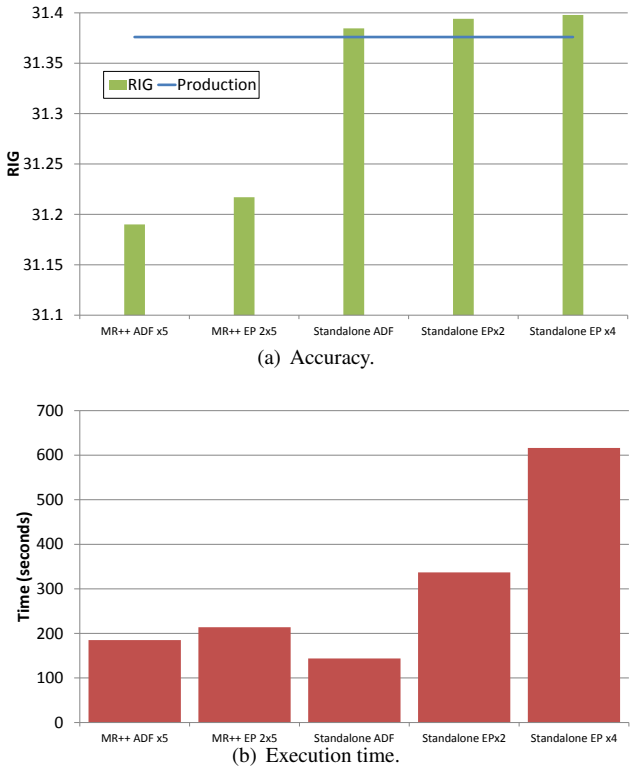

Figure 2. Accuracy and execution time for AdPredictor on 40 million click dataset.

Thus we have experimented extensively both with MapReduce and graph-based scale-out solutions for AdPredictor. A few years ago, scale-out was the only way to run such algorithms on real-world data sets. However, it is worth re-examining the need for scale-out in a world where machines can be "scaled up" to hundreds of gigabytes.

4. Evaluation

We evaluate the two versions of AdPredictor (EP and ADF) on a single server (Standalone) and on three cluster-based configurations: *MR Classic* (MapReduce), *MR++* (MapReduce with state retention), and *CamGraph*. For MapReduce and MapReduce++ we use CamDoop, a highly optimized implementation with considerably higher performance than Dryad or Hadoop [5]. Both CamDoop and CamGraph are implemented in C# and run on a customized cluster [4]. The cluster has 27 servers, each with two dual-core 2.66 GHz Xeon processors for a total of 4 cores (8 hyperthreads) per server. Each server has 12 GB of memory and an Intel X25 enterprise grade SSD. The click log data is stored on the SSDs and distributed uniformly over the 27 servers. For standalone we use a server with two 6-core 2.66 GHz Xeon processors for a total of 12 cores (24 hyperthreads) with 192 GB of memory on 6 DRAM channels in two NUMA domains. The click log data is stored on a single local SSD.

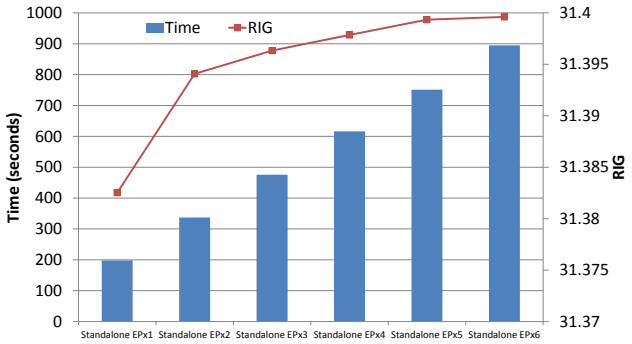

Figure 3. Comparison of execution time for AdPredictor on 40 million click dataset using single sever.

Figure 2 compares the MapReduce and Standalone versions of AdPredictor EP and ADF. We use 'xN' to denote N iterations (or MapReduce rounds). The MR++ EP configuration has nested iterations (local passes over the data as well as multiple MR rounds) and we denote this as 'MxN'. Figure 2(a) shows the RIG — the standard accuracy measure for this algorithm — for the MR++ and standalone configurations (MR++ is strictly better than MR classic in all cases, so we omit the latter). For reference we show the RIG achieved by the current production system using a custom implementation. Although the differences in prediction accuracy may appear small, they are significant as they apply to a large number of click-through predictions. We observe that ADF is faster than EP in both configurations but at the cost of additional complexity and loss in accuracy. Both MR++ configurations have significantly lower accuracy than standalone due to the additional approximation introduced by running a distributed MapReduce. Despite this, MR++ is slower than standalone for ADF and less than 2x faster than EPx2. MR++ runs on 27 servers whereas the standalone configurations are a single server running a single-threaded implementation.

Figure 3 shows the execution time and RIG of Standalone-EP with 1–6 rounds. Execution time is linear in the number of rounds, but accuracy flattens out beyond 2 rounds as the EP algorithm converges on a fixed point. Standalone EPx2 thus achieves near-ideal accuracy with competitive performance. Figure 4 summarizes all the results for AdPredictor. CamGraph retains the simplicity and accuracy of factor graphs and EP, but has poor performance due to the high network overheads of a partitioned graph algorithm. MapReduce shows better performance but sacrifices simplicity and accuracy. The Standalone configuration is simple, accurate, and within 2x of the performance on a 27-node cluster, simply by having more memory on a single server. Hence it would be fair to assume that the single big-memory machine performs about as well as 13.5 servers running MapReduce.

So far all the results we have presented use a relatively small data set of 40 million log entries, because CamGraph

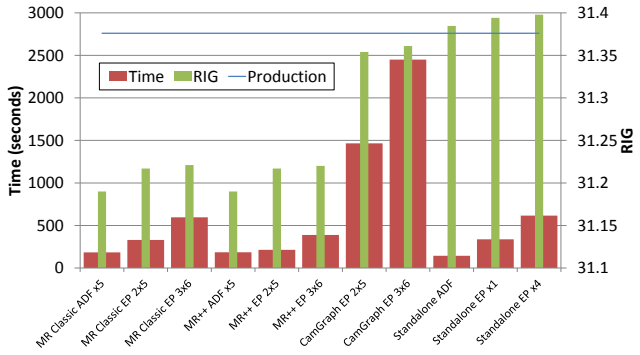

Figure 4. Comparison of performance and execution time for various versions of AdPredictor on 40 million click dataset using MapReduce (MR Classic), MapReduce++ (MR++), CamGraph and single server (Standalone).

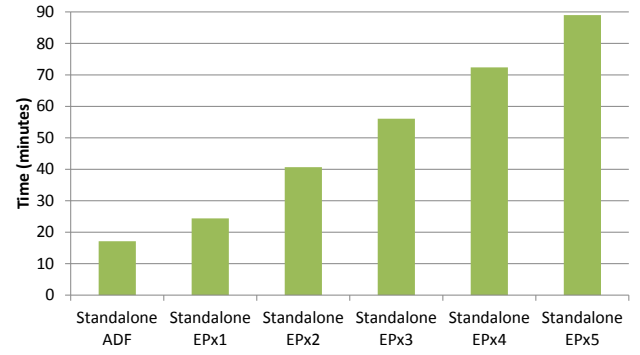

Figure 5. Comparison of execution time for various versions of AdPredictor on 280 million click dataset using single sever.

could not support larger data sets. However, the Standalone configuration with big memory does scale to larger data sets. In Figure 5 we show the time taken for ADF and EP with between 1 and 6 rounds for a 280 million click data set. The performance scales linearly with the data size, with Standalone EPx2 taking about 40 min.

Load time One potential concern with big-memory scaling is the time to read the input data from the network or storage device. The execution times shown so far include computation time and load time: the time taken to read and parse the input data. Figure 6 shows load time alone for the standalone ADF and EP configurations, for both the 40 million and 280 million click data sets. The 40 million entry click log is ~5 GB and the 280 million entry click log is ~34.5 GB. Load time for EP is slightly higher as it has to create a more complex data structure. Scaling to the larger dataset is slightly worse than linear: we believe this is due to the C# garbage collector. In all cases, the load time is significant, representing 20–25% of the total execution time. This suggests that as we scale up single servers by increasing memory sizes and core counts, we will need to simultaneously scale up I/O bandwidth to avoid this becoming a bottleneck.

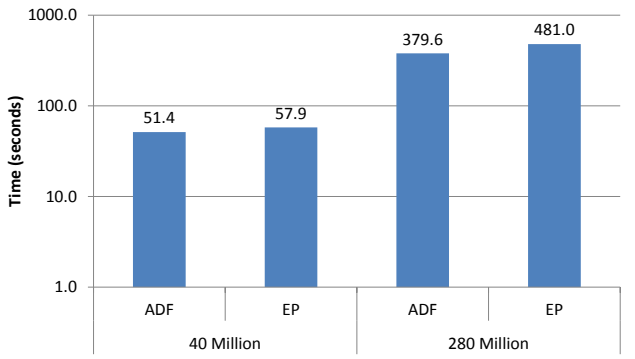

Figure 6. Comparison of load and data structure creation time for AdPredictor on 40 and 280 million click dataset using single sever.

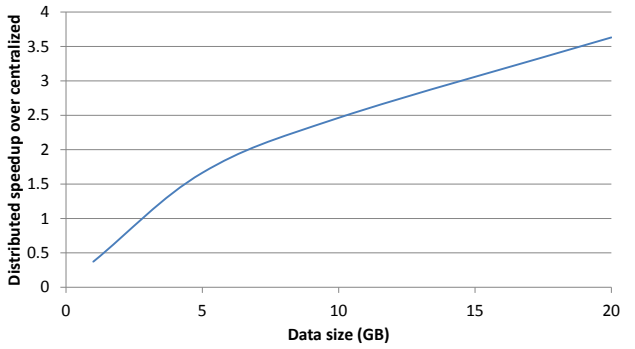

Figure 7. Comparison of MapReduce-like computation on a centralized server normalized by the performance of running distributed over 20 servers.

Ideal MapReduce workload AdPredictor is not an ideal fit for MapReduce. We repeated the comparison with a synthetic workload that is ideally suited to MapReduce: a simple aggregation over a key-value store. A large number of 64-bit keys with 350-byte values were uniformly partitioned over 20 servers. A local mapper on each server touches each value and extracts a 64-bit result; these are then reduced to a single 64-bit value per server by a machine-local reducer, and finally sent to a global reducer. The data were stored in a non-optimized in-memory key-value store written in C#. We compared the performance of this configuration with one in which all the data were stored on a single server.

Figure 7 shows the execution time for the single server normalized by the execution time across the cluster for data set sizes of 1.5 GB to 20 GB. At approximately 15 GB the cluster performs the operation three times faster than the single server. Hence, the 20 servers provide only 3 times the performance of the single server. Even for an "ideal" MapReduce job, the cluster performs 6 times worse per server and 4 times worse per dollar at these sizes.

Summary Scaling up with memory is a competitive option for AdPredictor. It reduces programmer effort and complexity, improves accuracy, and reduces hardware cost. At 40 million records, a single server performs only a factor of 2 worse than a cluster of 27 servers, yet achieves better accuracy. A non-optimized C# implementation on a 192 GB server can scale to 280 million records. The full production load could be run with 384 GB of DRAM, in about an hour using today's hardware and implementation, which we understand would be an acceptable frequency with which to run the job. Optimizing storage I/O would help to further reduce this time.

5. Conclusion and Future Work

Nobody (yet!) ever got fired for using a Hadoop cluster! But memory is becoming cheap, by historical trends very cheap, and this is potentially very disruptive. Single big-memory servers may simply be more efficient than clusters, which can substantially change price points and complexity. In this paper we have described the trends, and conjectured that many jobs simply do not need large-scale clusters to run. We have demonstrated this with a machine learning algorithm, where complexity and cost can be lowered by using a single server solution.

We have shown that AdPredictor could scale using a server with big memory rather than clusters. We have also experimented extensively with a second algorithm: Frequent Itemset Mining (FIM) [2]. This is used to determine sets of items that occur together frequently, e.g., in shopping baskets. This also has a version that is designed to support parallel or distributed implementation called SON [9], which also was suitably complex that the parallel version of the algorithm was published. The distributed version predates MapReduce, but is well suited to implementation on a MapReduce-like framework, and we have implemented on all the same platforms as AdPredictor. We omit the results here for space reasons, but they support similar conclusions as for AdPredictor.

Are these results applicable more generally? We believe that DRAM sizes are at a tipping point for human generated data. Examples of such data are social (e.g. Twitter, FourSquare) and shopping baskets. The size of such data is fundamentally limited by the number of people on the planet, which (fortunately) does not double every 18 months. Core counts and DRAM sizes per server by contrast, are still on a Moore's Law trajectory. Back of the envelope calculations convince us that a 512 GB server could process all the items purchased at a major UK food retailer in the last year or could handle a GPS location per day per person in the UK, for an entire year.

We also see some new challenges introduced by "big memory":

Storage I/O Data for analytics must be loaded before it can be analyzed, either from local storage or from a scalable storage back end, and this can become a bottleneck. If we follow the argument in this paper, then we think that we can afford to spend more on I/O bandwidth per server (as we will have significantly fewer servers) and we conjecture that we should aim at at least 2–4 10 Gbps Ethernet ports per server or 1–4 SSDs (at 300 MB/s each) if using local storage.

Storage filtering: Datasets such as click logs have many different analytics run on them, each potentially requiring only a few fields out of 10s or 100s. We need easy and efficient *filtering* of the data at or near the storage servers.

Programming: We need to think of providing programming abstractions and runtimes that support big-memory servers efficiently, but also allow programs to be transparently run on clusters when they truly need the scale beyond a single server.

So, to conclude, for workloads that are processing multi-gigabytes rather than terabyte+ scale, a big-memory server may well provide better performance per dollar than a cluster. When will Amazon rent me a 1TB VM?

Acknowledgments

We thank Thomas Borchert and Jurgen Van Gael for patiently teaching us about factor graphs, expectation propagation, AdPredictor and providing data. We thank Christos Gkantsidis the information about job sizes and Khaled Elmeleegy for sharing a draft version of his paper with us.

References

[1] Apache Hadoop. http://hadoop.apache.org/.

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *VLDB*, pages 487–499, 1994.

[3] G. Ananthanarayanan, A. Ghodsi, AndrewWang, D. Borthakur, S. Kandula, S. Shenker, and I. Stoica. PACMan: Coordinated memory caching for parallel jobs. In *NSDI*, Apr. 2012.

[4] P. Costa, A. Donnelly, G. O'Shea, and A. Rowstron. CamCube: A Key-based Data Center. Technical Report MSR TR-2010-74, 2010.

[5] P. Costa, A. Donnelly, A. Rowstron, and G. O'Shea. Camdoop: Exploiting in-network aggregation for big data applications. In *NSDI*, Apr. 2012.

[6] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *OSDI*, 2004.

[7] K. Elmeleegy. Piranha: Optimizing short jobs in hadoop. In *under submission*, 2012.

[8] T. Graepel, J. Q. Candela, T. Borchert, and R. Herbrich. Web-scale bayesian click-through rate prediction for sponsored search advertising in microsofts bing search engine. In *27th ICML*, June 2010.

[9] A. Savasere, E. Omiecinski, and S. B. Navathe. An efficient algorithm for mining association rules in large databases. In *VLDB*, pages 432–444, 1995.