Vol.2017-DBS-165 No.30
Vol.2017-IFAT-128 No.30
2017/9/20

On Performance Under Hotspots in Hadoop versus Bigdata Replay Platforms

Marat ZHANIKEEV[1,a)]

Abstract: The core idea behind Hadoop is to distribute both the data and user software on individual shards within the cluster. The Bigdata Replay method is drastically different in that it packs user software into batches on a single multicore machine and uses circuit emulation to maximize throughout when bringing data shards for replay. The effect from hotspots, defined as drastically higher access frequency to a small portion of (popular) data, is different in the two platforms. This paper models the difference numerically but in a relative form, which makes it possible to compare the two platforms.

Keywords: hotspots, bigdata processor, Hadoop, Bigdata Replay, performance analysis

1. Introduction

Before embarking on this journey, it is necessary to admit that it is truly difficult to measure performance of any distributed Bigdata processing engine including Hadoop, as well as to discover and debug its performance bottlenecks. The processing itself is distributed by design with control and data traffic among networked nodes following a non-trivial logic. So, instead of building a comprehensive performance model/map for Hadoop, majority of existing research stops at contributing a number of workloads which can be used by system administrators to compare their own performance to that of others in relative terms [13].

Part of this difficulty is due to *inconsistency* in how the system performs under the same exact workload at different times. Wild variations in performance are documented in academic literature [11]. The reasons for inconsistency are the same: the system is too complex to capture and explain all the dynamics in a consistent and reliable manner.

There is also the **superlinear effect** [8] when the system is scaled. Note that that effect itself does not really exist, but is simply an artifact of various software and hardware specifics such as fixed buffers, timeouts, etc. However, when the overall performance of the system is measured at different scales, it produces a curve which convexes up at the beginning and then follows the saturation trend further on. Compared with the linear/diagonal line, this creates a crossing point beyond which performance is much worse than its linear expectation. The paper in [8] refers to this area as *payback* for the super-linear period at the head of the curve. Note that portion of analysis data further in this paper exhibits partially superlinear features as well.

There is an academic viewpoint at performance. For example, there is a paper from the original creators of Hadoop on the limits of its performance [9]. There is also recent paper that compares MapReduce with the more recent Spark processing engine [7]. The paper pays special attention to the Resilient Distributed Database (RDD) as a major contributor to the improved performance of Spark (over Hadoop) for majority of processing targets. More importantly, [7] recognizes that many practical targets require *multiple passes* on Hadoop because its simple *key-value* approach to processing cannot accommodate them within the single pass. Clearly, multiple passes drastically increase the completion time for a given task. A broader discussion of the subject and new solutions can be found in [1]. Finally, there is research that attempts to optimize jobs on Hadoop [14].

Following on the RDD argument above [7], there is research that recommends to abandon Hadoop altogether when increased RAM and storage on a single machine can accommodate the intended scale of processing [10].

As far as performance bottlenecks are concerned, current literature recognizes only *CPU, RAM and storage* as having potential to become a bottleneck [7]. Conversely, **network performance** is never considered as a potential bottleneck. In fact, the disinterest towards network performance is expressed by the creators of Hadoop themselves [9], who state that only 30% of traffic is internal to Hadoop and 70% is open to clients, that is, hinting that one should no worry about this resource becoming the bottleneck. Network as performance bottleneck is not found in any of the major recent papers that analyze and improve

[1] School of Management, Tokyo University of Science
Fujimi 1-11-2, Chiyoda-ku, Tokyo, JAPAN 102-0071
[a)] maratishe@gmail.com

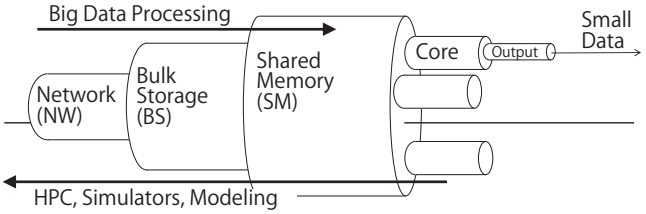

Fig. 1 All possible bottlenecks in bigdata processed regardless of the method. Incidentally, all the bottlenecks are shared with HPC methods.

performance on Hadoop and similar systems [7][8][9][10].

Note that network performance is part of Hadoop's internal operation, where it is found under the name of **Rack Awareness**[12]. The point is that system administrator assigns an arbitrary rack number to each *datanode*, while Hadoop uses this information to discriminate (and build preferences) between local (in-rack) and remote resources within the cluster.

There is a new method that brings a paradigm shift along with active role for network performance to bigdata processing, called the **Bigdata Replay** method (or simply Replay method further on) [1]. The core distinction (from Hadoop) of the Replay method is that processing code is not distributed but instead runs on a single multicore Replay Node. Data shards are bulk-transferred to the Replay Node using emulated circuits [2] where they are replayed while granting shared access to raw data to multiple processing jobs (running on the multicore).

This paper raises a novel viewpoint at performance analysis. First, the focus is mostly on **hotspots** [5] defined as drastically higher access frequency on a relatively small portion of the entire data bulk. This creates highly irregular access patterns [4], where the *irregularity problem* has recently been recognized by research community in relation to the topic of bigdata networking. Performance in this paper is limited to network which is suitable when comparing Hadoop to Replay methods. However, the analysis method itself generalizable and can be applied to any other resource including CPU, RAM and storage as is discussed in [7].

2. Performance in Hadoop versus Replay Methods

For the basics on Hadoop versus Replay methods see the short description above and full details in [1]. This chapter will focus on performance bottlenecks while comparing the two methods to each other.

Performance bottlenecks in a bigdata processing systems are described in Fig.1. For now, let us disregard the similarity between bigdata processing and High Performance Computing (HPC) and focus on the bigdata direction in the figure, which is left-to-right. The network is the narrowest bottleneck, followed by bulk storage (disks, HDD, SSD), then RAM (shared memory). Cores on multicore are not fully part of the model as they have little effect on the overall performance below the limits set by the preceding bottlenecks.

So, the key questions here is: while network is the narrowest bottleneck in Fig.1, why is it not part of performance analysis in academic literature on the subject? Refer to [7][8][9][10] to see the evidence supporting this argument.

There are two reasons why network is not traditionally considered to be a bottleneck. First, bottlenecks in CPU, RAM or storage are reached first on Hadoop simply because majority of time is spent on datanode where the job performs local calculations while networking happens only once at the end [7]. Secondly, performance analysis of the entire system would incorporate network performance but is difficult in practice due to the high complexity of the system.

Nevertheless, some attempts to incorporate network performance into analysis have been made. For example, [10] offers circumstantial proof that network bottleneck is important by removing the networking itself (single machine, no distribution) and showing that the system performs better without it. Moreover, network performance is part of the reason behind the superlinear effect [8], specifically the payback part of the curve.

The primitive argument made by the Replay method is the same as in [10], which is that the system's overall performance can be improved if a coarser unit of distribution is used. The method itself in [10] is the coarsest possible form which is to remove distribution completely.

Replay method [1] uses a cruder unit of processing. In Hadoop, the unit is source code running on each shard, per client. Under the Replay method, the source code is running on multicore and shares access to a shard with other jobs. This paper will not delve into details on job packing and its optimization but [1] and more recently [4] can offer the necessary details on the subject.

The Replay method is stronger in *irregular* environments as well. Jobs in batches work in the same data, using the same region of shared memory, is a stabilizing factor, compared to the jitter introduced by distribution over the network. However, even under the Replay method, jobs have some jitter and experience skew in processing time over time, which is resolved using dynamic job re-packing [4].

The analysis in this paper focuses on the weakest link in Fig.1 which is **network performance**. Incidentally, it is the easiest way to compare the two methods. In Hadoop, in-job traffic (Reduce phase of the MapReduce, for example) depends on network performance. In Replay, the bulk transfer of the shard itself is limited by the bottleneck. The focus of analysis in this paper is on hotspots, which are defined as some shards accessed disproportionally more frequently than others. Since jobs in the Replay method are packed by the respective data shard, performance dynamics of the method are expected to be different.

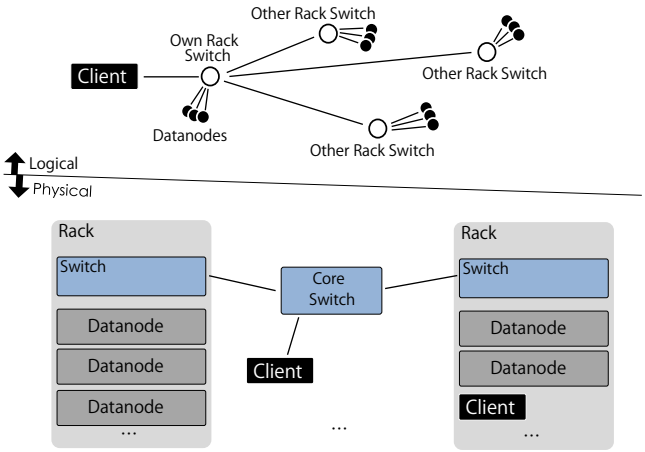

Fig. 2 Rack Awareness feature in Hadoop versus its irrelevance when modeling the system from the viewpoint of a client.

3. Rack Awareness and Performance Bottlenecks

Repeating an earlier statement, Hadoop incorporates network performance in form of the **Rack Awareness** feature [12] which is part of the majority of its internal functionality. For example, the first replica of a newly added data is placed *in-rack* (using Hadoop terminology [12]). When processing data, the job selects shards on in-rack datanodes as much as possible, unless they too crowded by other jobs. Note that this functionality directly relates to the concept of *hotspots* in the intended analysis. The downside to the Rack Awareness feature on Hadoop is that it can only be configured manually, which hints that automatic assignment background monitoring of networking performance is not offered by the system. The alternative would be to build and maintain a logical graph based on *network distance* (expressed in delay, throughput, etc.) – see [3] for more pointers on network distance.

The lower part of Fig.2 shows how Hadoop views the system using the Rack Awareness feature. Since in Hadoop all nodes (including clients) are expected to be part of a rack, the system assumes that each client (the source of data, in this case) has a number of datanodes which are much closer to it (in terms of network performance) than the rest of datanodes. Note that, some clients can be connected to core switch, in which case their rack affiliation is ambiguous.

The upper part of Fig.2 shows that rack awareness is irrelevant as far as performance bottleneck is concerned. Regardless of rack affiliation, there is always the *closest switch*. Even *in-rack* datanodes are on the other side of the in-rack switch (unless special connectivity is used), although network performance on connections to them should be relatively better. By comparison, other datanodes are behind another additional network hop which are switches in other racks in the cluster. The core switch in Fig.2 can also easily become the bottleneck unless a **multidimensional interconnect** is used by the cluster. However, even with a multidimensional interconnect, there

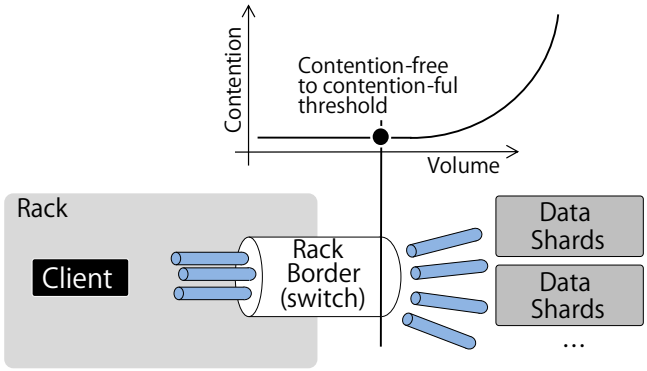

Fig. 3 Modeling used for performance optimization in this paper.

is always an upper cap on capacity simply because all-to-all connectivity across racks is infeasible in practice. See more on interconnect capacity optimization in [3].

Fig.3 simplifies the view and focuses on performance analysis. The client is connected to the outside world through the bottleneck, which is the network switch in this case, but can be other resources (CPU, RAM, storage). The bottleneck (even network) is not a single resource but is an aggregate of multiple unit resources, which are physical lines in the specific case of Fig.3.

The **contention** specifics behind Fig.3 are as follows. Contrary to the traditional viewpoint on contention [8] which views it as a continuous function/curve, this paper assumes that contention is of the type of the *load-to-response curve* in [6]. Up to a given point it is assumed that that system is *contention-free*. However, past a given threshold, contention intensifies sharply followed by drastic impairment in performance. Note that this feature is commonly observed in majority of congested systems today such as Content Delivery Networks (CDN), resource virtualization [6], etc.

Also note that such a viewpoint allows us to use the curve as a binary (congested or not) *threshold* in analysis further in this paper.

4. Basics of Hotspot-y Workload Generation

This paper models irregularity using the *hotspot distribution*. Details about the generation of a hotspot distribution can be found in [5] – the paper applies it to generation of realistic synthetic workloads. The hotspots themselves represent **popularity** or, more academically speaking, *access frequency* of a given data shard. This section explains the general concept and the method used to apply the hotspot curves in analysis.

The hotspot model is based on four sets of numbers referred to as *normal*, *population*, *hot* and *flash*. In reality, there are only three sets as *hot* and *flash* describe the same items at two different stages in their lifespans. The sets can be used to describe a wide range of heterogeneous phenomena occurring in nature.

In complex modeling, the process is modeled in time, allowing items to grow from *hot* to *flash* gradually – the case

of Flash Crowd, viral, and other similar events. However, for the purposes of modeling in this paper, the simplified version from [5] is applied, meaning that only the static sets are used without any time dynamics between them.

Yet, even in the simplified *sets-only* form, distributions are difficult to judge. It is helpful to classify distributions based on their curvature. The following classification method is applied in this paper for the first time. First, imagine the log values of the sets plotted in decreasing order of value. Given the nature of the distribution, *hotspots* – there are normally only a few of them – would be plotted at the head of the distribution and then the curve would drop for the rest of values. Note that the drop would be experienced even on the log scale. Here, the only way to classify such a distribution is to judge the size of its *head*.

So, classification in this paper uses the following ranges for classification, all in log scale:

- if values at 80% and further into the list are 0.15 or above, then *Class A* is assigned;
- if values at 60% and further into the list are 0.6 or above, then *Class B* is assigned;
- if values at 40% and further into the list are 1.3 or above, then *Class C* is assigned;
- if values at 30% and further into the list are 1.8 or above, then *Class D* is assigned;
- if no class is assigned by this point, the *Class E* is assigned.

The class assignment above is fine-tuned to the distributions used for analysis further in this paper. The tuning was done in such a way that each class would get a roughly equal share of distributions, which were otherwise spread over a reasonably large parameter space. As another simplification, each hotspot distribution is converted into two separate curves by combining *popular + hot* and *popular + flash* set, each classified and used separately. Only 100 values were generated for each distribution, but this is sufficient as values during simulation are selected randomly from the list, which means that relatively low values are selected much more frequently than the large items. Otherwise, the same process as in [5] is used, where there are 2-3 other parameters such as variance, number of hot items, variance across hot items, etc.

Fig.4 shows several curves from the dataset, all marked in accordance with its calculated class. We can see that the classification is successful by assigning a higher letter to curves with a relatively higher number of hotspots. Note that the generation process has no maximum value, but, in order to avoid extremely long processing sessions, all values exceeding 1000 are curtailed to 1000. The values (not logs, but the original number behind) are used in simulation to define a relative difference in access frequency.

5. Analysis Model and Setup

This section puts in numbers the modeling method explained in Sections 2 and 3 above. Let us set shard size as S. Immediately, we can distinguish between Hadoop and Replay methods by using the ratio of *in-job traffic to shard size* itself, denoted as r (expecting a fraction of 1). Here Hadoop generates rS traffic per shard, while the Replay method bulk-transfers the entire shard for replay and therefore generates traffic volume of S.

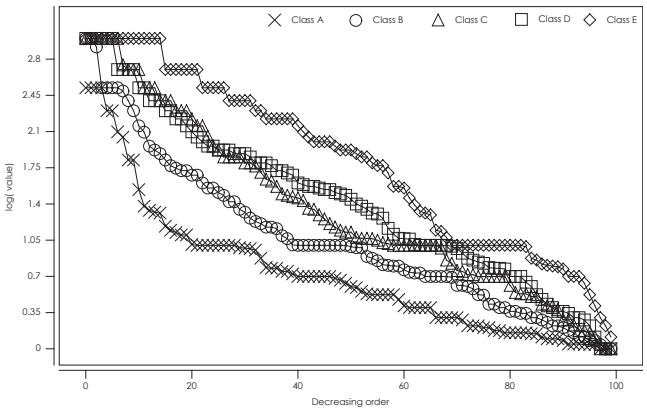

Fig. 4 Example distribution each attributed to a class from A to E. Vertical scale is in log.

Based on the modeling simplification explained in Section 3, let us denote *contention threshold* as C – incidentally the symbol is often used for *capacity* which, in these settings, is a nearly identical concept.

Hotspots are treated differently in the two methods. Under Hadoop, each client generates its own separate processing job, which is another way to say that *hotspots are disregarded*. Under the Replay method, client jobs are grouped in batches based on the requested shard. In fact, taking the CDN viewpoint and assuming that only top 5-10% of shards classified as hotspots (in fact, this is the entire point of the hotspot distribution), it is not too much of a stretch to require the Replay Node to cache all hot shards locally. This would mean that bulk transfer for each hot shard would take place only once. However, this *technological leap* is left for future publications on the subject, while this paper assumes that jobs are packed into batches based on a fixed time interval (the value 10 is used further in analysis). This means that shards are unique (have to be bulk-transferred) not only by their id but also by the (rounded up) request time.

So, having the list of shard hotness (popularity) $\{h_1, h_2, h_3, ..., h_n\}$ and separately the list of shard sizes $\{S_1, S_2, S_3, ..., S_n\}$, we can calculate traffic generated by each method. For Hadoop the traffic volume is

$$V_{hadoop} = \sum_{i=1..n} rh_i S_i. \quad (1)$$

For the Replay method it is:

$$V_{replay} = \sum_{i=1..n} S_i. \quad (2)$$

Note that Hadoop normally uses the fixed shard size but for the sake of argument let us keep the sizes separately in the generic form. Also note that we cannot right away tell which traffic volume is larger as it depends on the setting of r, the class of hotspot distribution, etc.

Let us perform another minor simplification before going into the analysis proper. By setting all shard sizes to $S = 1$ and setting the value of C as a multiplier of this unit S, we can define the *time till contention* performance metric which is defined as the number of (parallel) jobs it takes to take the bottleneck over its contention threshold.

This also simplifies comparison between the two methods. One simply has to replay a hotspot distribution in simulation and see which method gets to a given C faster, where *faster* is indicative of *poorer* performance, comparatively speaking. The replay process is extremely simple, as the simulation simply picks shards probabilistically from the selected hotspot distribution. Hot items are selected proportionally more frequently.

6. Analysis Results

Fig.5 presents the matrix of performance plots for all combinations of *hotspot class*, r and C parameters, where the *time till contention* metric on the vertical axis of each plot is the outcome. For hotspot class, values are *A through E*, representing gradually increasing class (see Section 4 for details on hotspot classes). The number in legends in plots is for the given value of r which is randomly selected from $\{0.001, 0.005, 0.01, 0.05, 0.1, 0.2, 0.5\}$ in each simulation run. The outcome is aggregated in each plot for a distinct combination of hotspot class (columns) and value or r (vertical). Performance itself is shown as the *area* covering all the performance outcomes encountered during the 100k simulation runs necessary to sufficiency cover all permutations of parameter values.

The short conclusion from Fig.5 is that the Replay method wins the performance race. Hadoop surpasses the Replay method for $r \geq 0.05$ (meaning 5% of in-job traffic volume) but this effect, like the *superlinear* effect in [8], is a misreading coming from the fact that hot items generate too many jobs at once and *overshoot* the threshold by a large margin. In the long run (which in the plots is the larger values for C), this effect is counteracted by the longer horizon for contention, in which case the Replay method always shows superior performance.

Fig.5 also shows that performance for the Replay method has higher scattering of results, which depends mostly on the class of the hotspot distribution. By comparison, scattering under Hadoop decreases with growing C. Note that even with larger scattering, Hadoop stays strictly below the Replay area (on the log scale).

7. Conclusions

This paper is takes a rare viewpoint at performance of bigdata processors by focusing on performance under hotspots. Granted Hadoop is indifferent to them as each job gets its own independent set of resources (regardless of shard popularity), there are other methods which can benefit from taking shard popularity into consideration. The specific other method discussed in this paper is the Bigdata Replay method which replays shards in a space shared by multiple parallel jobs.

The outcome of this paper, in a manner of speaking, contradicts intuition. On one hand, assuming that in-job traffic (Reduce stage in MapReduce) is very small compared to the total shard volume, one can expect Hadoop to have better performance by default. However, analysis results show that having multiple jobs work on the same data in parallel offers performance improvement sufficient to make the Replay method more efficient, by comparison. Note that in analysis in this paper, jobs were packed with the ration of 10-per-shard (temporal packing), yet even with the ratio of in-job traffic set to 0.0001 of total shard volume, the Replay method outperformed Hadoop.

There is more room for performance improvement on the part of the Replay Method. Popular (hot) shards can be cached locally which would further decrease bulk traffic. Future publications will pursue this among other potential values for performance improvement under the Replay method.

References

[1] M.Zhanikeev, “Streaming Algorithms for Big Data Processing on Multicore”, Big Data: Algorithms, Analytics, and Applications, ISBN 978-1-4822-4055-9, CRC, pp.215–240, February 2015.
[2] M.Zhanikeev, “Circuit Emulation for Big Data Transfers in Clouds”, Networking for Big Data, CRC, ISBN 978-1482263497, pp.359–392, September 2015.
[3] M.Zhanikeev, “The Switchboard Optimization Problem and Heuristics for Cut-Through Networking”, The 23rd IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN), Osaka, Japan, June 2017.
[4] M.Zhanikeev, “Volume and Irregularity Effects on Massively Multicore Packet Processors”, Network Operations and Management Symposium (APNOMS), Kanazawa, Japan, October 2016.
[5] M.Zhanikeev and Y.Tanaka, “Popularity-Based Modeling of Flash Events in Synthetic Packet Traces”, IEICE Technical Report on Communication Quality, Vol.112, No.288, pp.1–6, November 2012.
[6] M.Zhanikeev, “Optimizing Virtual Machine Migration for Energy-Efficient Clouds”, IEICE Transactions on Communications, vol.E97-B, no.2, pp.450–458, February 2014.
[7] J.Shi, J.Qiu, U.Minhas, L.Jiao, C.Wang, B.Reinwald, F.Ozcan, “Clash of the Titans: MapReduce vs. Spark for Large Scale Data Analytics”, 41st International Conference on Very Large Data Bases, Kohala Coast, Hawaii, USA, pp.2110–2121, September 2015.
[8] N.Gunther, P.Puglia, K.Tomasette, “Hadoop Superlinear Scalability”, ACM Queue, vol.13, issue 5, May 2015.
[9] K.Shvachko, “HDFS scalability: the limits to growth”, Usenix Login, vol.35, number 2, April 2010.
[10] A.Rowstron, D.Narayanan, “Nobody ever got fired for using Hadoop on a cluster”, 1st International Workshop on Hot Topics in Cloud Data Processing (HotCDP), April 2012.
[11] M.Xia, N.Zhu, Y.He, S.Elnikety, “Performance Inconsistency in Large Scale Data Processing Clusters”, 10th USENIX International Conference on Autonomic Computing (ICAC), San Jose, CA, USA, pp.297–302, June 2013.
[12] Hadoop: Rack Awareness. [Online]. Available: https://hadoop.apache.org (August 2017)
[13] Big Data Benchmark. [Online]. Available: https://amplab.cs.berkeley.edu/benchmark (January 2017)
[14] A.Rasooli, D.Down, “COSHH: A Classification and Optimization based Scheduler for Heterogeneous Hadoop Systems”, Future Generation Computer Systems, vol.36, pp.1–15, July 2014.

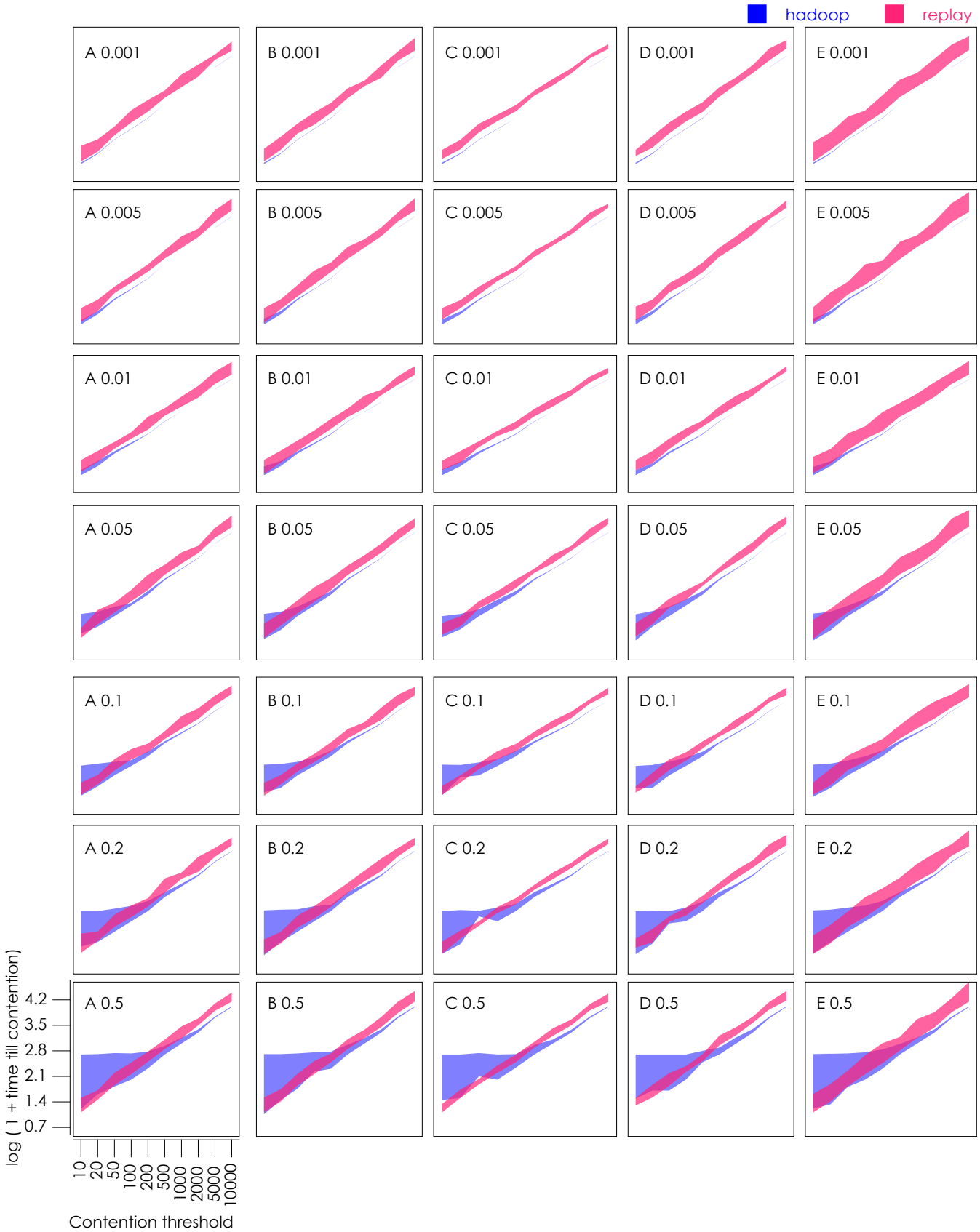

Fig. 5 Relative comparison of performance between Hadoop and Replay methods. Legend on each plot shows its *hotspot class* and r setting. Both horizontal and vertical scales are (roughly) in log scale, which means that the linear trends in plots are, in fact, hyper-exponential in reality.