Read-performance of ZABBIX with NoSQL databases and HistoryGluon

MIRACLE LINUX CORPORATION*

August 13, 2013

Abstract

This paper presents the benchmark result of read performance for ZABBIX modified to write and read history data in a NoSQL DB. The results show that read throughput and latency with Cassandra via ZABBIX API are approximately the same level as those without the modification in the actual large-scale monitoring environment. This means that ZABBIX with Cassandra increases the maximum number of monitoring targets and monitoring frequency without degradation of read performance by a combination of our previous work that enhances write performance with Cassandra.

1 Introduction

We reported research progress for improving write performance of ZABBIX [1] with NoSQL DBs [2] in February, 2013. The approach in the research saves history data in a NoSQL DB. Because history data is the major part of written data, ZABBIX with NoSQL increases the maximum number of monitoring targets and monitoring frequency. Although this is a useful finding, read performance also has to be practicable to use it in actual operations. ZABBIX reads history data mainly for showing graphs and a response of `history.get` ZABBIX API request.

We improved ZABBIX so that it can read history data stored in a NoSQL DB by the our previous enhancement [2]. This paper describes the benchmark method and the results.

2 Benchmark method and the setup

2.1 Benchmark method

We prepare two connection modes to read history data as shown in Figure 1. One is 'Front-end connection mode' that reads history data via ZABBIX API. This corresponds to the actual read path. The other is 'Direct connection mode' that reads history data directly from Databases: MySQL, PostgreSQL, and NoSQL(Cassandra). The benchmark for the latter is almost the equivalent to that for databases themselves. However, the result is useful to see overhead of ZABBIX Front-end that mainly serves ZABBIX API.

In both modes, HistoryGluon [3], an abstraction interface for NoSQL DBs, is used. It was originally developed in our previous study [2]. We added functions for reading history data in a NoSQL DB to HistoryGluon server and HistoryGluon PHP extension module that is used in ZABBIX Front-end.

`zabbix-benchmark` in the figure is a benchmark script we developed. It is also open source software [4] and is implemented by Ruby. It uses zbxapi library [5] to get data via ZABBIX API in Front-end connection mode. In case of Direct connection mode, it uses RDBMS adapters (mysql2 [6] and pg [7]) and HistoryGluon Ruby binding included in HistoryGluon [3].

2.2 Benchmark scheme and the setup

We measure read throughput and read latency as indexes of performance. The combinations of databases and connection modes on the benchmark are listed in Table 1. We only use Cassandra[8] as a NoSQL database, because the write performance with Cassandra is significantly better than that with the default DB (MySQL and PostgreSQL) in our previous benchmark [2].

We also evaluate the effect of parameters listed in Table 2. When the read operation is used for graph drawing, the requested duration is various depending on the viewer's purpose. Therefore, evaluating its dependency is important. As the number of monitoring targets increase, the amount of data ZABBIX has to store is almost

*The lead author: Kazuhiro Yamato, e-mail: kazuhiro.yamato@miraclelinux.com

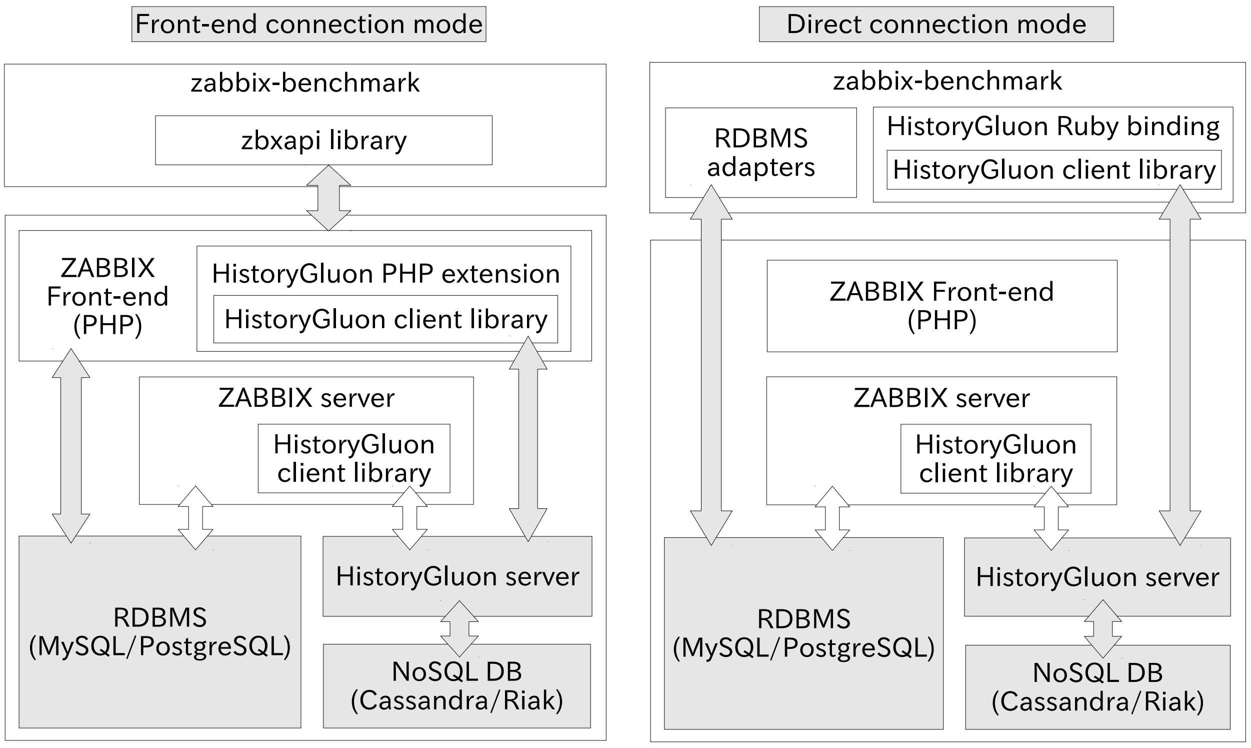

Figure 1: Software stacks for our read performance benchmark.

Table 1: Combinations of databases and connection modes.

Connection mode	DB (other than history)	DB (history)	Available w/o modification of ZABBIX
Front-end	MySQL	MySQL	Yes
Front-end	MySQL	Cassandra	No
Front-end	PostgreSQL	PostgreSQL	Yes
Front-end	PostgreSQL	Cassandra	No
Direct	MySQL	MySQL	Yes
Direct	MySQL	Cassandra	No
Direct	PostgreSQL	PostgreSQL	Yes
Direct	PostgreSQL	Cassandra	No

proportional to it. In general, ZABBIX server keeps writing history data in the actual situation. Hence we also check the dependency on it. For the number of monitoring target: 90, they are only for ZABBIX server itself and the monitoring interval (T_{int}) is the default (60 to 3600 seconds). This condition is nearly equal to the situation without writing.

The monitoring interval for other cases (almost all items in the condition with the number of monitoring target: 5610 and 27690) is 5 seconds. This interval is somewhat greater than the typical value in the actual situation. It is to create a condition corresponding to a system with a lot of monitored hosts. When the actual interval is 5 minutes, the conditions with the number of target: 5610 and 27690 are comparable to that with 3600 and 18000 monitored hosts respectively.

Throughput and latency are measured by `zabbix-bencmakrk` script that performs the following steps.

1. Register monitoring targets with ZABBIX API.

2. Save test history data for 100 days in a history database on the ZABBIX server. The number of items for one target hosts is 92. The items are based on 'Template OS Linux' and consist of various types such as an integer, a floating point, and a string.

3. Read history data from one monitoring host for 120 seconds repeatedly when throughput is measured. The read item is randomly selected every time. Finally the number of the read operation the script performed is reported.

Table 2: Parameters used in the benchmark.

Parameter	Value
Requested duration (days)	10, 20, 30, 40, 50, 60, 70, 80, 90, and 100
The number of monitoring targets (items)	90, 5610, and 27690 (correspond to 0, 3600, and 18000 hosts when $T_{int} = 5$ min.)

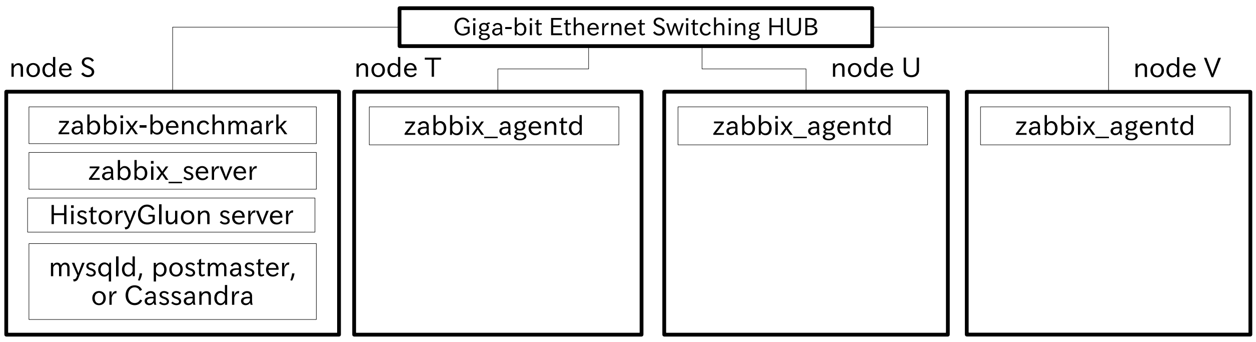

Figure 2: A configuration of machines used in the benchmark.

or

Read 10 items and calculate the average when read latency is measured. The read item is randomly selected every time.

Figure 2 shows the configuration of machines. All machines have the same specification as listed in Table 3. The benchmark script and other major software components are executed on one machine. Three machines for zabbix_agentd are concurrently used to generate a lot of history data.

3 Results and discussion

3.1 Read throughput

Figure 3, 4, and 5 show read throughput versus the requested history duration. The marker types indicate the combinations of databases and the connection modes in Table 1. The open and filled markers mean the conditions with RDBMS only and with NoSQL (Cassandra) respectively.

On all results, throughput increased as the history duration. This is considered that the efficiency of the read was improved due to the increase of the number of items per one request. In addition, results with Direct connection mode were greater than those of the same database combination with Front-end connection mode. This implies that ZABBIX Font-end has a certain overhead.

When the number of targets is 90 items (Figure 3), throughput of `MySQL only (Direct)` and `PostgreSQL only (Direct)` were roughly the same and approximately twice as high as those with Cassandra. This characteristic was the same for throughput with Front-end connection mode.

When the number of targets is 5610 items (Figure 4), throughput of `PostgreSQL only (Direct)` was the best. Compared to throughput with 90 items, it decreased where the requested history duration $\geq$ 80

Table 3: Specification of machine and software used in the benchmark.

CPU	Intel Core i5-3570K 3.40GHz
Chipset	Intel Z77 Express
Memory	DDR3 1600MHz 4GB $\times 4$
HDD	Seagate ST2000DM001 formatted with EXT4 (No LVM is used)
LAN	Realtek RTL8111/8168B PCI Express Gigabit Ethernet controller
Distro.	CentOS 6.3 (64bit)
Java	OpenJDK 1.7

days. The ratio is approximately 0.7 to 0.8. Throughput of `MySQL + Cassandra (Direct)` and `PostgreSQL + Cassandra (Direct)` were the next. They had little degradation from the conditions with 90 items and were approximately half as high as that of `PostgreSQL only (Direct)`. Throughput of `MySQL only (Direct)` was greatly decreased by approximately 0.15 to 0.2 compared to that with 90 items. It was almost the same as other results with Front-end connection mode and was approximately one third compared to that of `PostgreSQL only (Direct)`.

When the number of targets is 27690 items (Figure 5), throughput of `MySQL + Cassandra (Direct)` was the highest and decreased by approximately 0.7 compared to that with 90. Throughput of `PostgreSQL only (Direct)` is the second best result and was approximately 0.6 as high as that of `MySQL + Cassandra (Direct)`. Throughput of `MySQL only (Direct)`, `MySQL only (Front-end)`, `MySQL + Cassandra (Front-end)`, and `PostgreSQL + Cassandra (Front-end)` were the next better and approximately 0.4 as high as that of `MySQL + Cassandra (Direct)`. `PostgreSQL only (Front-end)` and `PostgreSQL + Cassandra (Front-end)` had the worst result in this condition. Their throughput were approximately 0.2 as high as that of `MySQL + Cassandra (Direct)`.

3.2 The read latency

Figure 6, 7, and 8 show read latency versus the requested history duration. The meaning of marker types is identical to that in section 3.1. We can see the trend in which the latency increased as the history duration.

When the number of targets is 90 items (Figure 6), the results of combinations without Cassandra were roughly equal to or shorter than those with Cassandra. However, the relation was dissolved, as the number of targets increased (Figure 7 and 8). Generally, the latency with the combination that had the good result in terms of throughput was also good (short). For the results with Front-end connection, there was not big difference between the combination with and without Cassandra (NoSQL) under the condition with a certain write operations (Figure 7 and 8).

4 Summary

We improved ZABBIX [1] so that it can also read history data stored in a NoSQL DB by the our previous enhancement [2]. The benchmark results with the tool [4] showed that read throughput and latency in Front-end connection mode were roughly the same level both with and without Cassandra (NoSQL) except the condition with 90 items in which the write frequncey is close to zero (i.e. no hosts are monitored). Thus we confirmed that ZABBIX with Cassandra (NoSQL) increases the maximum number of monitoring targets and monitoring frequency without degradation of read performance in the actual large-scale monitoring environment.

References

[1] http://www.zabbix.com

[2] http://www.miraclelinux.com/jp/online-service/labs/pdf/zabbix-write-performance/at_download/file

[3] https://github.com/miraclelinux/HistoryGluon

[4] https://github.com/miraclelinux/zabbix-benchmark

[5] https://github.com/red-tux/zbxapi

[6] https://github.com/brianmario/mysql2

[7] http://deveiate.org/code/pg/

[8] http://cassandra.apache.org

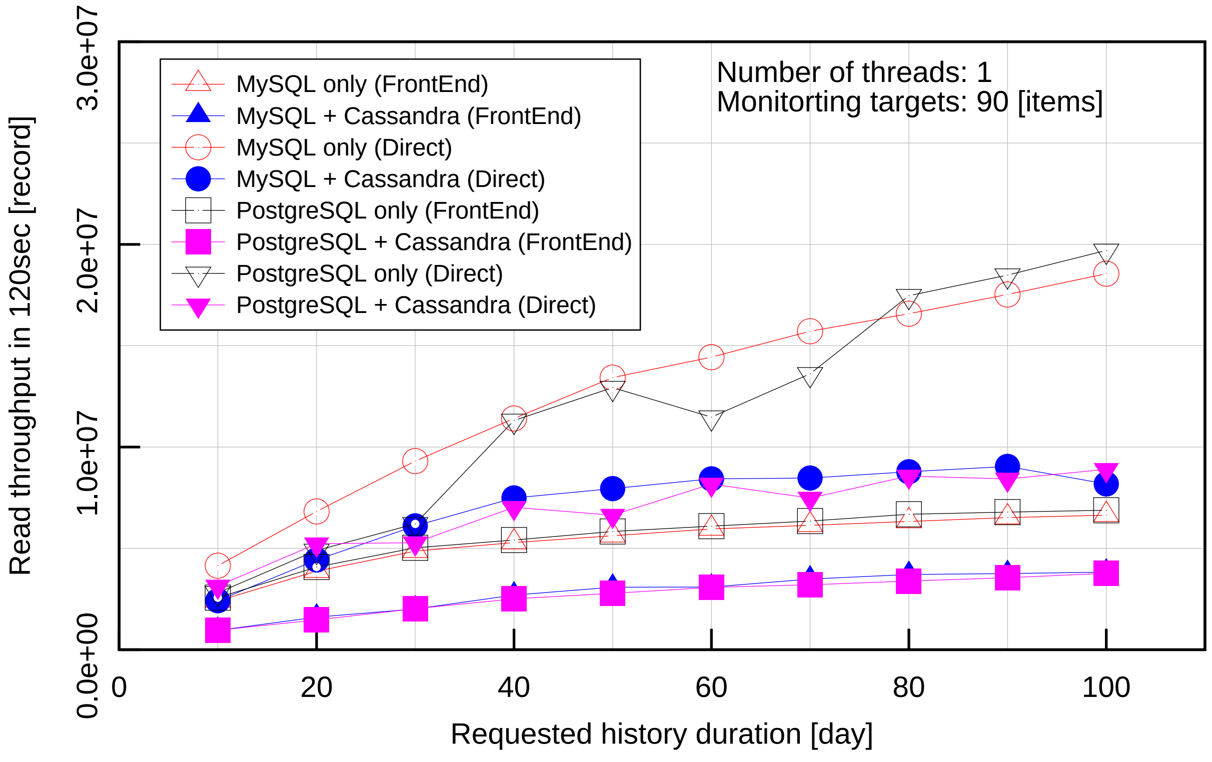

Figure 3: Read throughput (90 monitoring targets)

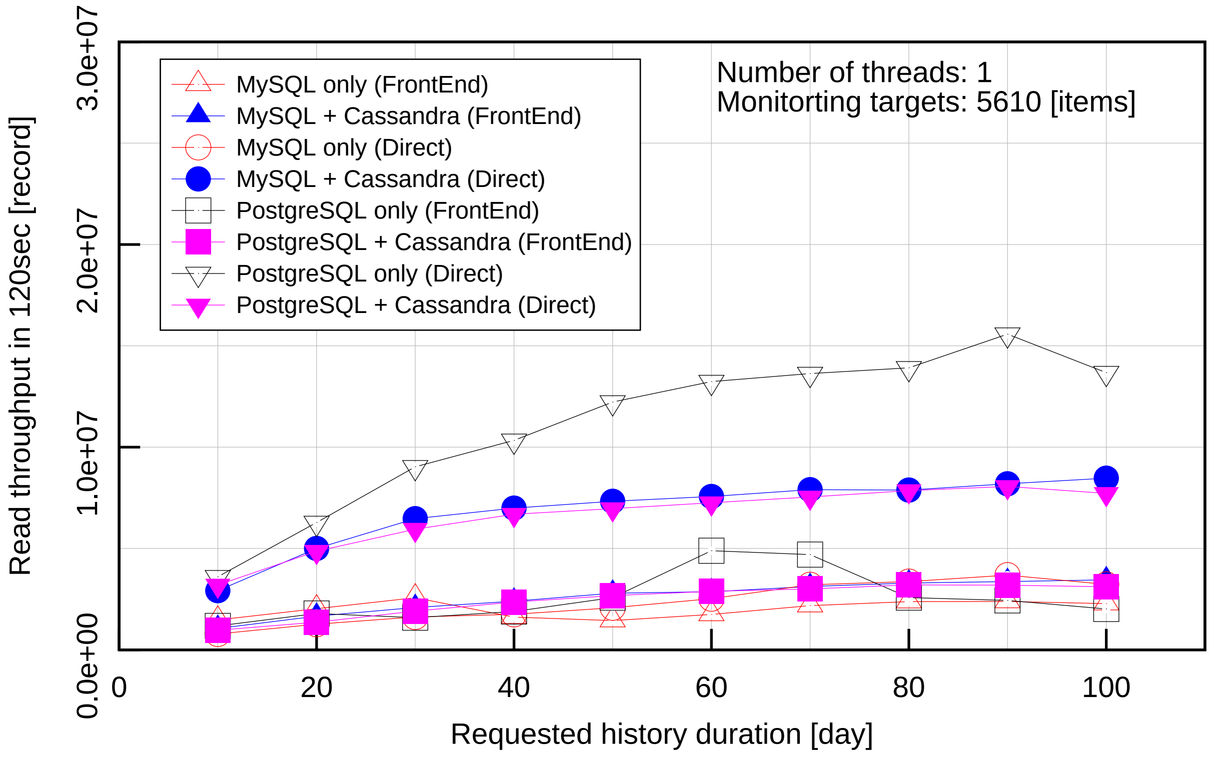

Figure 4: Read throughput (5610 monitoring targets)

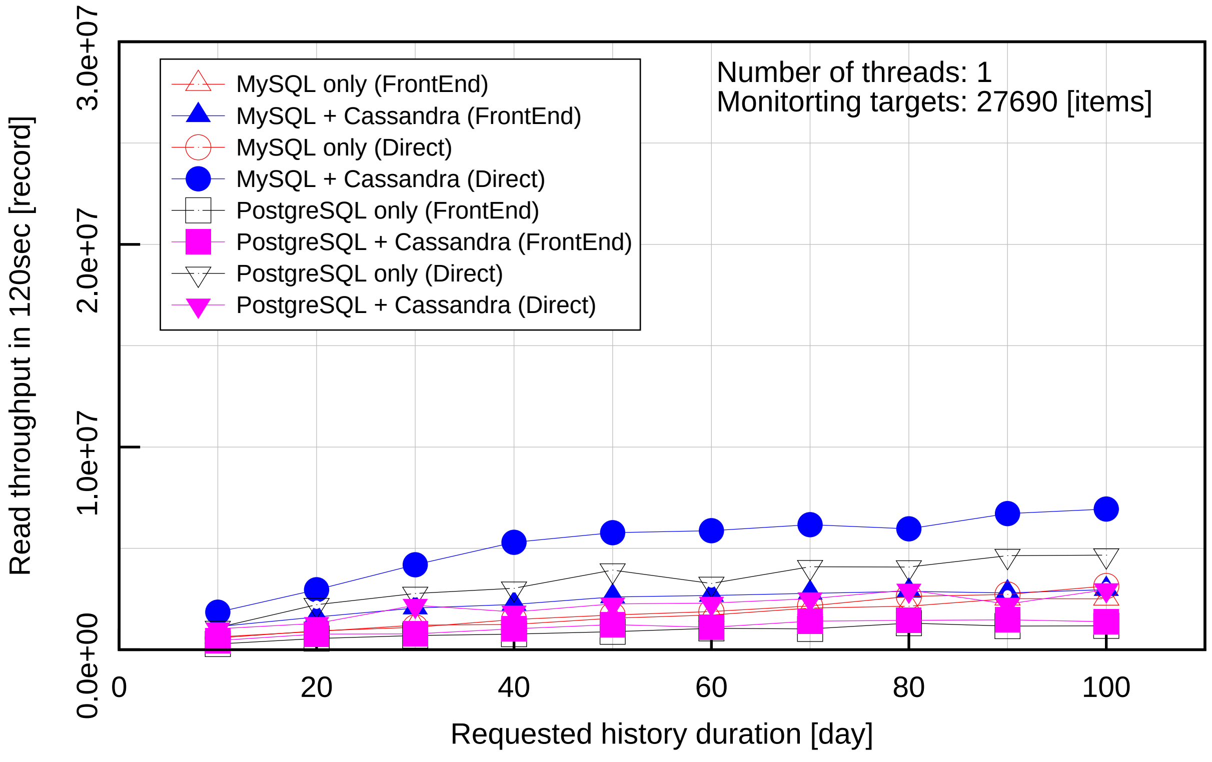

Figure 5: Read throughput (27690 monitoring targets)

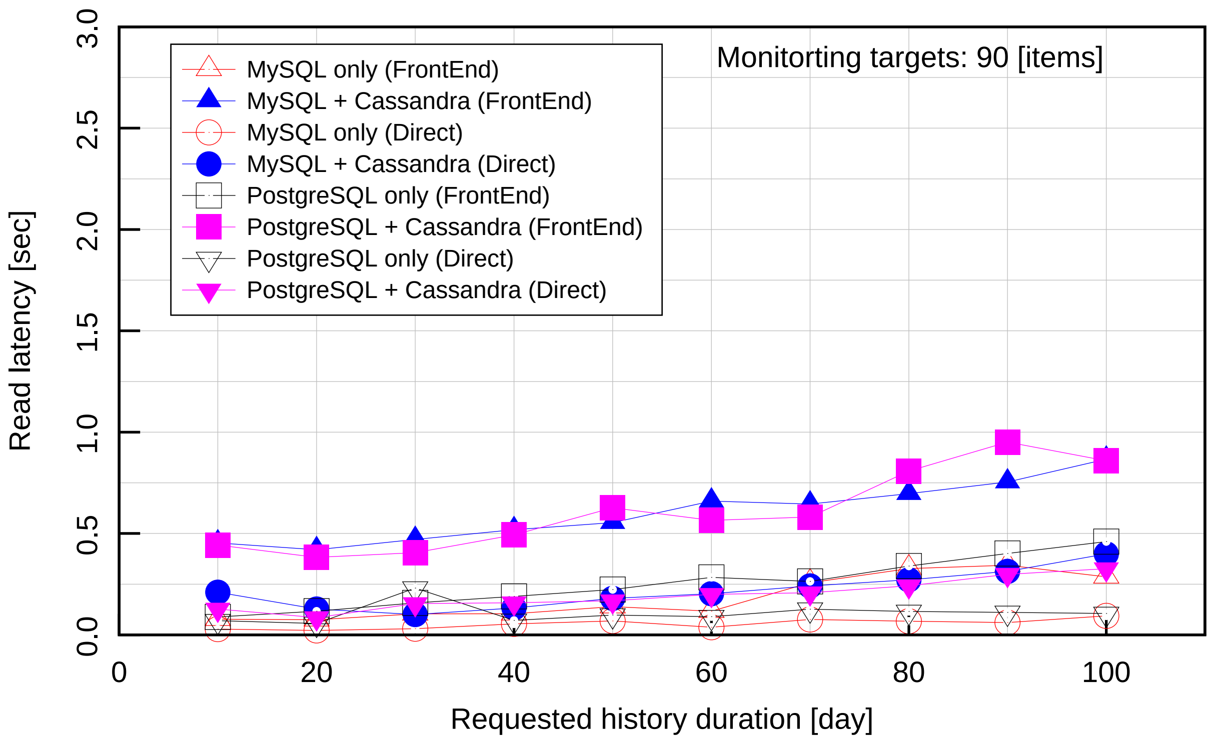

Figure 6: Read latency (90 monitoring targets)

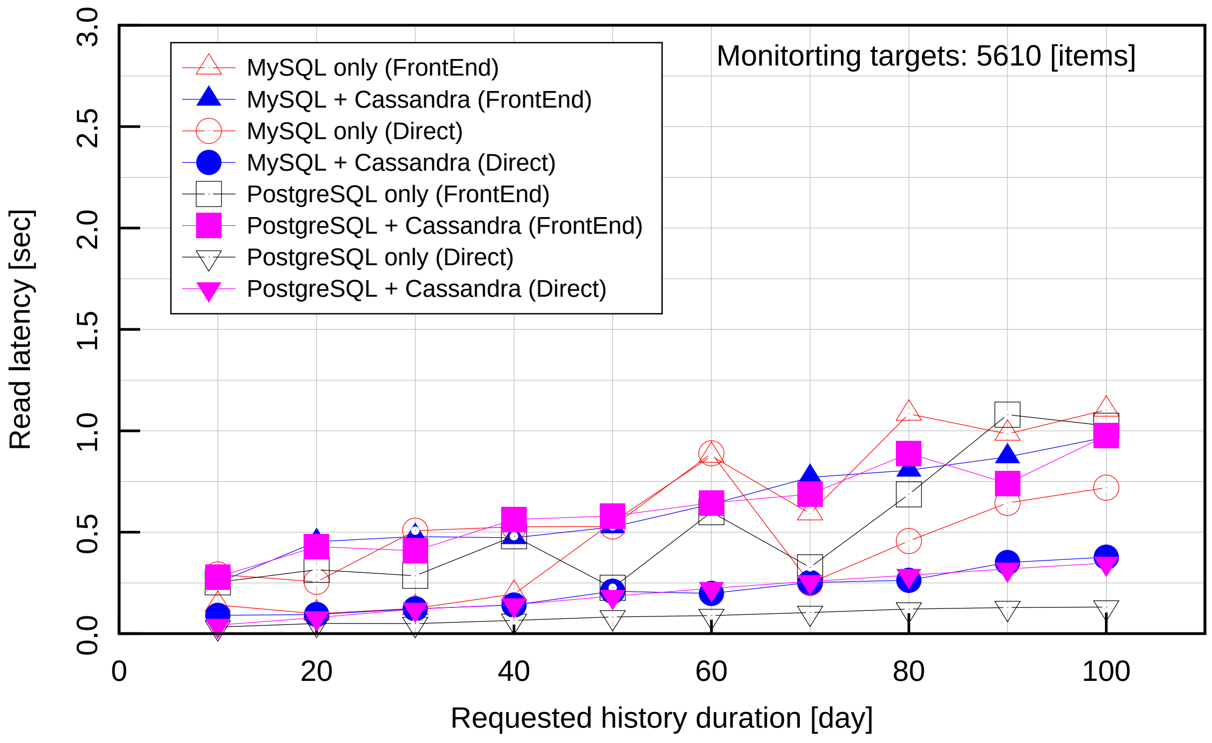

Figure 7: Read latency (5610 monitoring targets)

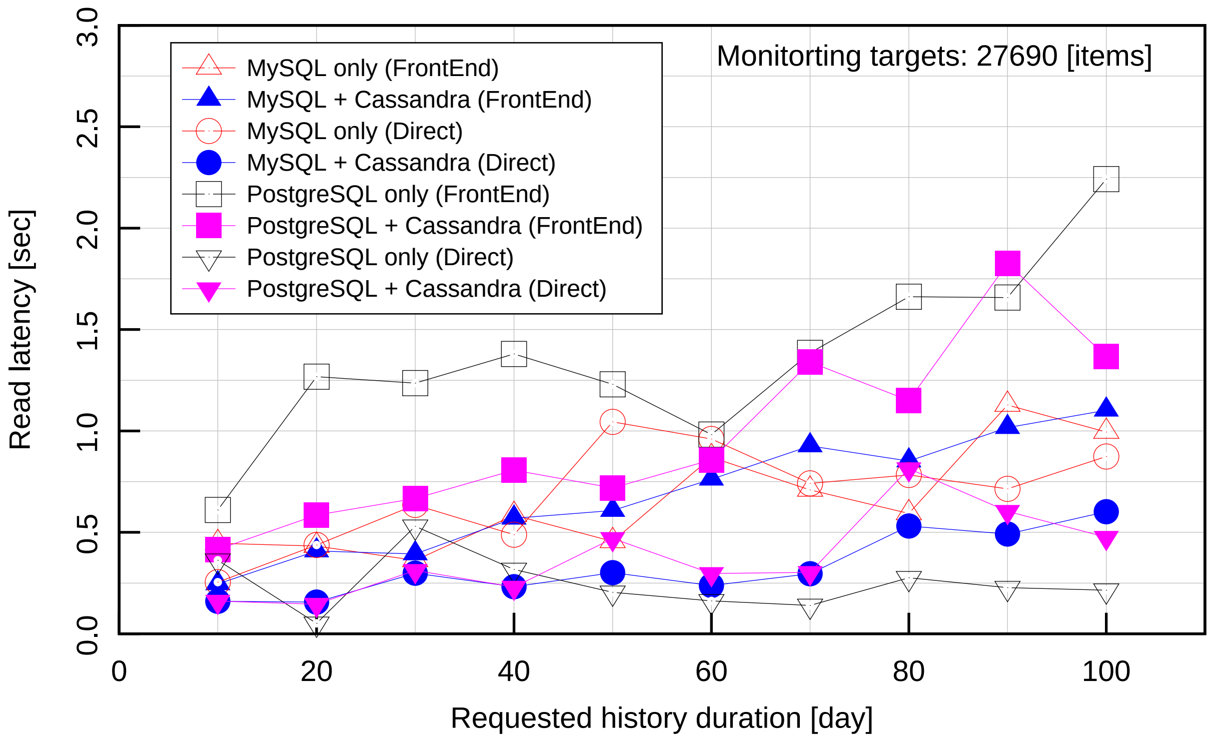

Figure 8: Read latency (27690 monitoring targets)