influxdata

Assessing Write Performance of InfluxDB Clusters using Amazon Web Services

AN INFLUXDATA TECHNICAL PAPER

Todd Persen
VP of Engineering, InfluxData
Robert Winslow
External Contributor

December 2016

Overview

In this technical paper, we'll explore the aspects of scaling clusters of InfluxDB with our closed-source InfluxEnterprise product, primarily through the lens of write performance. As a horizontally-scalable product with myriad use-case-dependent options, it can be difficult to give a single answer that indicates how InfluxEnterprise behaves across different configurations.

This data should prove valuable to developers and architects evaluating the suitability of InfluxEnterprise for their use case, in addition to helping establish some rough guidelines for what those users should expect in terms of write performance in a real-world environment.

It should be noted that we chose to run these tests exclusively on top of Amazon Web Services, using the EC2 and EBS services that we see our customers using most often. Obviously, performance characteristics will differ if these tests are run on dedicated hardware or even on virtualized instances from other cloud vendors, but we felt that standardizing on the most prevalent platform would be most beneficial to our community.

Our goal with this benchmarking test was to create a consistent, up-to-date comparison that reflects the latest developments in InfluxDB and InfluxEnterprise. Periodically, we'll re-run these benchmarks and update this document with our findings. All of the code for these benchmarks are available on GitHub. Feel free to open up issues or pull requests on that repository or if you have any questions, comments, or suggestions.

What is Time-Series Data?

Time-series data is nothing more than a sequence of values, typically consisting of successive measurements made from the same source over a time interval. Put another way, if you were to plot your values on a graph, one of your axes would always be time. For example, time-series data may be produced by sensors like weather stations or RFIDs, IT infrastructure components like apps, servers, and network switches or by stock trading systems.

Time-series databases are optimized for the collecting, storing, retrieving and processing of time series data; nothing more, nothing less. Compare this to document databases optimized for storing JSON documents, search databases optimized for full-text searches or traditional relational databases optimized for the tabular storage of related data in rows and columns.

Baron Schwartz has outlined some of the typical characteristics of a

About InfluxDB

InfluxEnterprise Version Tested: v1.1.0

InfluxDB is an open-source time-series database written in Go. At its core is a custom-built storage engine called the Time-Structured Merge (TSM) Tree, which is optimized for time series data. Controlled by a custom SQL-like query language named InfluxQL, InfluxDB provides out-of-the-box support for mathematical and statistical functions across time ranges and is perfect for custom monitoring and metrics collection, real-time analytics, plus IoT and sensor data workloads.

InfluxEnterprise is the closed-source version of InfluxDB, which offers high-availability, redundancy, and horizontal scalability via the clustering feature, designed for on-premise deployments. This is ideal for production deployments where data integrity, fault tolerance, and operational resiliency are required.

For more information about InfluxEnterprise, visit influxdata.com.

purpose-built time-series database include:

- 90+% of the database's workload is a high-volume of high-frequency writes
- Writes are typically appends to existing measurements over time
- These writes are typically done in a sequential order, for example: every second or every minute
- If a time-series database gets constrained for resources, it is typically because it is I/O bound
- Updates to correct or modify individual values already written are rare
- Deleting data is almost always done across large time ranges (days, months or years), rarely if ever to a specific point
- Queries issued to the database are typically sequential per-series, in some sort of sort order with perhaps a time-based operator or function applied
- Issuing queries that perform concurrent reads or reads of multiple series are common

Summary

For this initial phase of benchmarking, we chose to focus only on write throughput. Most use cases we've examined generally experience far greater write load than query load, so we felt that focusing on the parameters involved in scaling the write path to meet requirements would likely result in a cluster capable of handling a normal query load as well.

In building this benchmark suite, we identified a few parameters that are most relevant to scaling write performance. As we'll describe in additional detail below, we looked at performance across three vectors:

1. **Number of Data Nodes**
2. **Replication Factor**
3. **Batch Size**

The trends for these are relatively straightforward. We expect throughput to increase with more data nodes and a larger batch size, and to decrease with a higher replication factor (since the replication factor means that data has to be written throughout the cluster multiple times).

Additionally, we used varying numbers of workers during our benchmarks to ensure that we were actually able to generate and send a large enough data volume to truly saturate the clusters during the tests.

The Data Set

For this benchmark, we focused on a dataset that models a common DevOps monitoring and metrics use case, where a fleet of servers are periodically reporting system and application metrics at a regular time interval. We sampled 100 values across 9 subsystems (CPU, memory, disk, disk I/O, kernel, network, Redis, PostgreSQL, and Nginx) every 10 seconds. For the key comparisons, we looked at a dataset that represents 10,000 servers over a 24-hour period, which represents a decent-sized production deployment. We also provided some color about how these comparisons scale with a larger dataset, both in duration and number of servers.

Overview of the parameters for the sample dataset

Number of Servers	10,000
Values measured per Server	100
Measurement Interval	10s
Dataset duration(s)	24h
Total values in dataset	8,640,000,000 / day

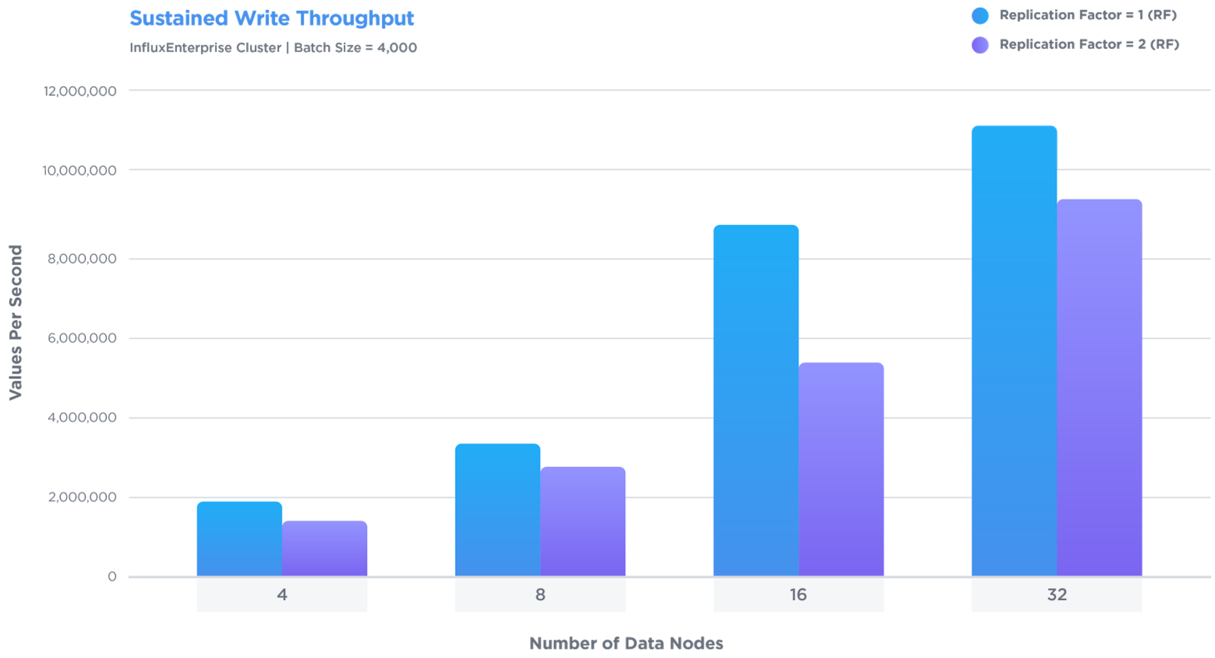
This is only a subset of the entire benchmark suite, but it's a representative example. At the end of this paper we'll discuss other variables and their impacts on performance. If you're interested in additional detail, you can read more about the testing methodology on GitHub.

Note that, to keep benchmarking times reasonable across many different runs, we performed bulk loads using fixed time limits. This means that the entirety of the data set was not always loaded to completion.

Cluster Size and Replication Factor

When looking at rates of ingestion for clusters, there are two primary factors that have the greatest impact on overall throughput - the **number of data nodes** and the **replication factor**. In general, we recommend using an RF=2 for any workload that requires redundancy or high-availability. Using a higher replication factor could be useful if you require it for additional resiliency, and a replication factor of one might be appropriate for maximizing throughput in situations where data loss is acceptable.

Below is a graph that shows the tradeoffs between varying replication factors in a cluster with 4 data nodes and with workers sending batches of 4,000 points per request. (Insert Sheet #1)

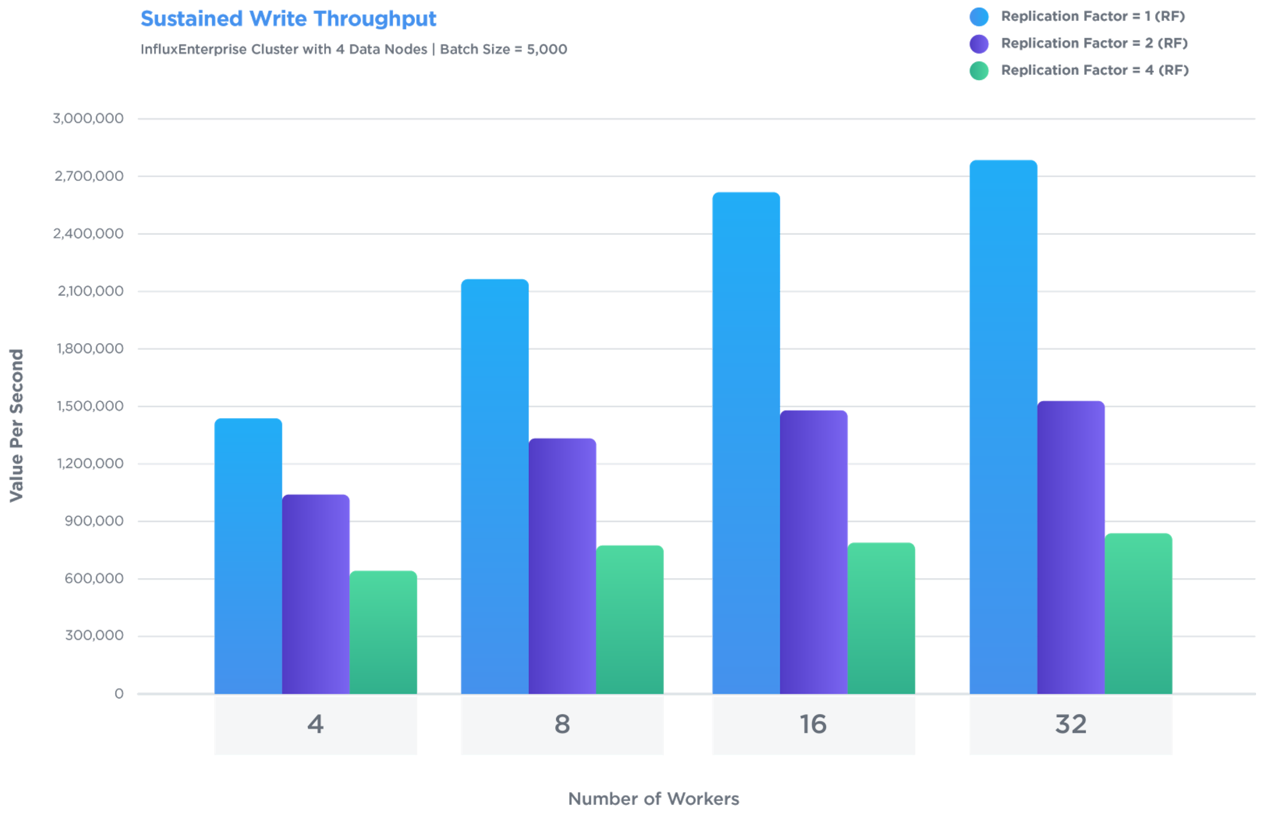


As you can see, there is an strong positive correlation between throughput and the number of data nodes, which is desirable as a facet of horizontal scalability in a data platform. There may obviously be a theoretical point where network cross-talk will limit the effectiveness of this scale-out, but these results indicate that a 32-node cluster is still unhindered by those second-order effects.

CONCLUSION:

> ***InfluxEnterprise scales horizontally for writes, showing throughput increases for up to 32 data nodes.***

Additionally, as we built out our tests, we experimented with various numbers of workers sending data to see what level of concurrency was required to saturate the write throughput of the cluster. The graph below shows the effects of increasing the number of workers in a cluster with 4 data nodes, with each worker sending batches of 5,000 points. (Insert Sheet #3)



As the graph shows, you get better throughput using more workers writing data, with the highest measured configuration here employing 8 workers per data node. Obviously, this will be workload-dependent, as the workers in our benchmarks post each batch sequentially, as quickly as possible.

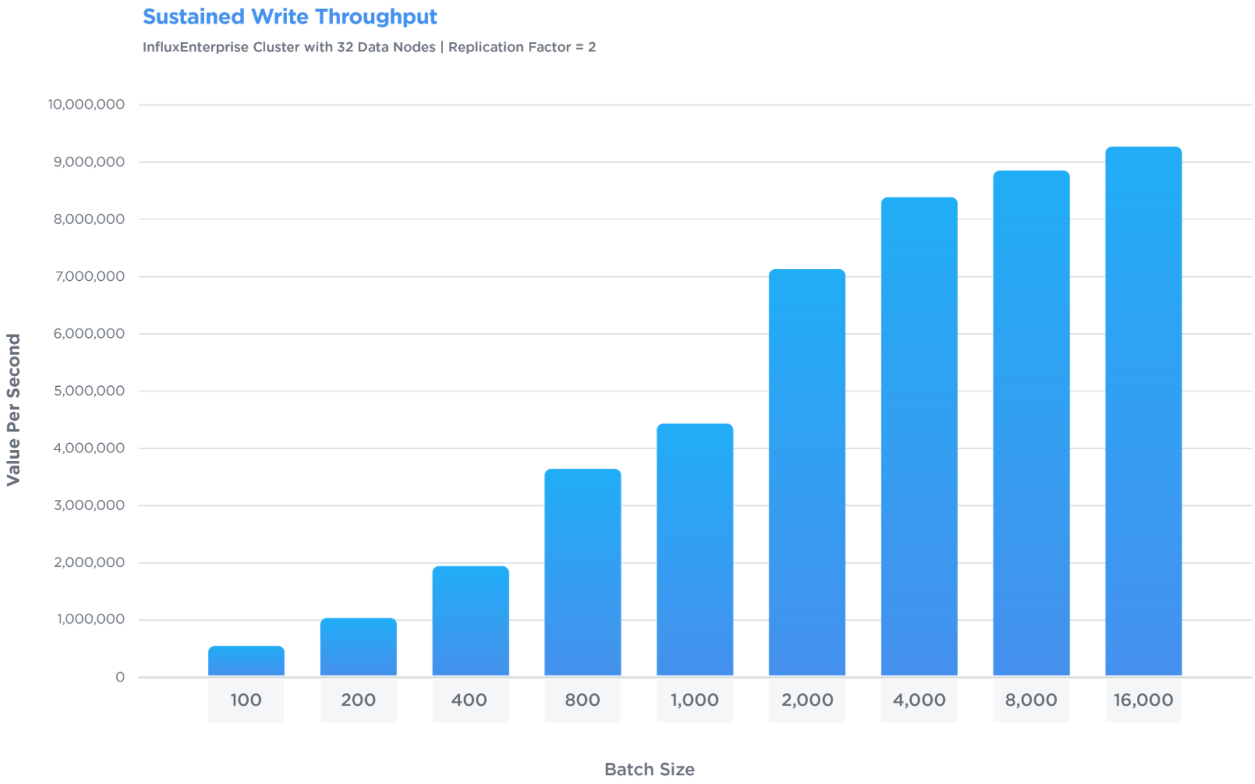
CONCLUSION:

> ***For maximum write performance, make sure you use multiple concurrent workers per data node.***

Batch Size

The final parameter that has a significant impact on write performance is **batch size**. When talking about batch size, we're referring to the number of values sent in a single request to the InfluxDB cluster. Since there is significant network and HTTP overhead with each request, in addition to per-request database processing, larger batches tend to be far more efficient to process overall.

In the chart below, you can see that overall write throughput increases consistently as batch size increases. In general, we recommend aiming for batch sizes of 5,000 to 10,000. You can see that the effectiveness of increasing the batch size is decreasing as we approach the 8,000 and 16,000 batch sizes. (Insert Sheet #2)



In practice, the downside of extremely large batch sizes is that the individual payloads become quite large, are slower to transfer over the network, and require more resources to process at the server. We recommend testing larger batch sizes and to choose the highest batch size that gives you significantly better throughput than the previous size. In other words, if you double the batch size and only see a 5% increase in throughput, you have probably reached the point of diminishing returns and should stick with the previous batch size.

CONCLUSION:

> ***Using larger batch sizes will generally increase total write throughput to the cluster.***

Testing Hardware

All of the tests performed were conducted on the Amazon Web Services EC2 platform, using **m4.2xlarge** instances for the data nodes and a single **t1.micro** meta node, using 32GB GP2 EBS volumes for storing the WAL, TSM data, and Hinted Handoff queues. All nodes were provisioned inside of a single VPC.

Summary

In the course of this benchmarking paper we looked at the write performance of InfluxEnterprse across three vectors:

- Cluster Size
- Replication Factor
- Batch Size

The benchmarking tests and resulting data demonstrated that along these vectors, InfluxEnterprise should generally:

- Achieve higher write throughput with more data nodes
- Achieve lower write throughput with a higher replication factor
- Achieve higher write throughput with larger batch sizes

Most excitingly, we demonstrated that a 32-node cluster of InfluxEnterprise is capable of handling over **11.1 million** writes per second singly-replicated, and over **9.3 million** writes per second doubly-replicated. In contrast, single nodes of InfluxDB generally hit peak performance at just under 1 million writes per second, even on the most performant hardware.

We highly encourage developers and architects to run these benchmarks themselves to independently verify the results on their hardware and data sets of choice. However, for those looking for a valid starting point on which settings will help optimize write performance, these results should serve as a useful guide for scaling and optimizing InfluxEnterprise clusters.

What’s Next?

InfluxDB Documentation, Downloads & Guides

- 1.1 Download
- 1.1 Installation Guide
- 1.1 Getting Started
- 1.1 Schema Design
- 1.1 Line Protocol Reference
- 1.1 Key Concepts

Benchmarking Resources

- Benchmarking Code, Methodology and Documentation on GitHub

Have Questions or Need Help?

- InfluxDB Google Group
- contact@influxdata.com
- Technical Support
- Public Training
- Private Training
- Virtual Training