Measuring Apache Ignite Scalability

Group 4: Asli Senel, João Pedro Henriques Pereira, and Lukas Spiekermann

Instituto Superior Técnico, University of Lisbon Lisbon, PORTUGAL

1 Introduction

The following is a list of the people who are involved in this project:

- Lukas Spiekermann, lukas.spiekermann@tecnico.ulisboa.pt, 105008.
- João Pedro Henriques Pereira, joao.h.pereira@tecnico.ulisboa.pt, 93587.
- Asli Senel, asli.senel@tecnico.ulisboa.pt, 105017.

The git commit identifier (SHA) to be evaluated is

cd963291e93bdac7f8a8efbc181553736d7c27a9.

This report will analyse the scalability for a distributed database. Distributed database systems built a system structure that provides access to requesting programs and users via an interface. As if it were a matter of access to only one database. Apache ignite's database uses RAM as the default storage and processing plane. It is an open source platform for distributed in memory databases, caching and real-time transactional data computation of large data sets [8].

Today it is important to be able to adapt to constantly changing requirements/environment more easily. We only depend on scalability that is the reason why we choose this project - databases must be scalable. After looking through multiple option of systems we finally decided to go with Apache Ignite, because it's supposed to be highly scalable, the variety of functionalities it provides which can be used for a lot of cases and the support it gives to the server and client.

Afterwards we continue with the system description. Followed by chapter 3 where the experiment-setup and results are described. We then conclude the report with a discussion of the main insights obtained.

2 System Description

Apache Ignite is an open source in-memory distributed database and computing platform. In this chapter we will present Ignite's main characteristics. More details of the characteristics can be found in the Apache Ignite documentation [6].

- Durable Memory and Ignite Persistence – Ignite treats RAM as a complete functional storage layer, meaning you can turn the persistence on and off. If persistence is turned off it acts as a distributed in-memory database or

in-memory data grid, depending if you use SQL or key-value APIs. If it is enabled, Ignite becomes a distributed, horizontally scalable database that guarantees consistency, making it so that there is no need to keep all data and indexes in memory because Durable Memory is tightly coupled with persistence and treats it as a secondary memory tier, and also supports full cluster restarts.

- ACID Compliance – Ignite is a strongly consistent system because data stored in Ignite is ACID-compliant, atomicity, consistency, isolation and durability, both in memory and disk.
- Complete SQL Support – Ignite has a full support for SQL, DDL and DML, and allows users to use pure SQL, having such SQL support makes Ignite a very good distributed SQL database.
- Key-Value – The in-memory data grind makes Ignite a fully distributed key-value store that scales horizontally up to hundreds of servers.
- Co-located Processing – Ignite allows collocating computations with data making it so Ignite minimizes data movement.
- Scalability and Durability – Ignite supports adding and remove cluster nodes, also allows storing multiples copies of data, making it resilient to partial cluster fails.

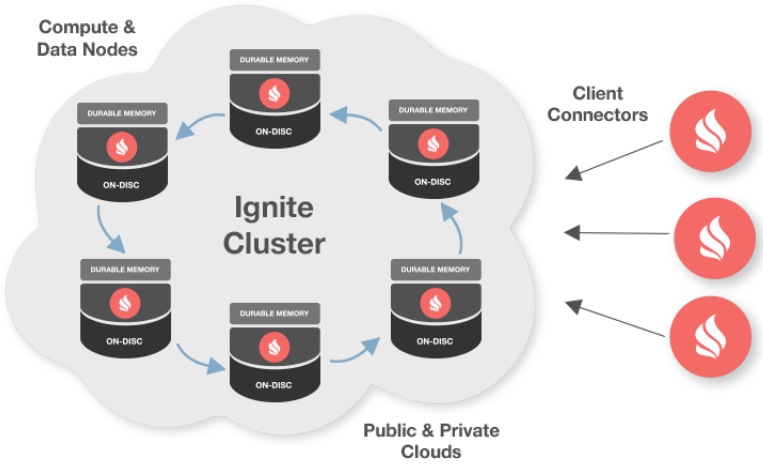

Fig. 1. Ignite advanced cluster [6]

When Apache Ignite is deployed as a data grid, the application layer begins to treat Ignite as the primary store. As application write and read from that data grid, Ignite ensures that all external databases stay updated and consistent. [3]

Ignite automatically load balances jobs produced as well as individual tasks submitted via computing API. Also uses off-heap memory to allocate memory regions outside Java heap, but can be enable making it in-heap caching. Making it useful in scenarios when you do a lot of cache read on server nodes or invoke cache entries deserialization.

Has for the nodes inside the cluster each one of them has a copy of the cache that contains certain parts of the data in .bin and .dat files. Data partitioning is the method the subdivides large sets of data into smaller chunks and distributes them through all nodes in a balanced manner, when the number of nodes changes it goes through a process called rebalancing where this partitions are distributed [1].

Apache Ignite includes 2 modes of cache operations that affect the nodes, partitioned and replicated, while partitioned splits partitions equally between all server nodes making it more scalable, replicated is when all data is replicated through every node in the cluster providing most availability of data.

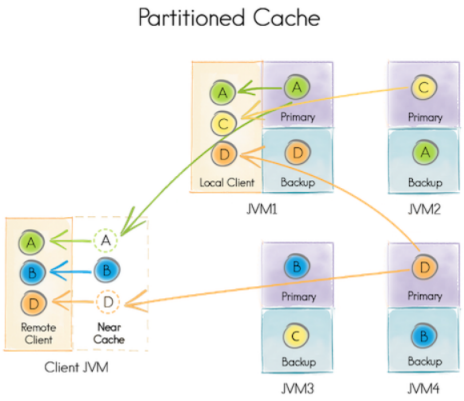

Fig. 2. Partioned Mode example [1]

Having a look into the the pipeline we can see that the requests are sent to the cluster that contains 1 or mode nodes that have access to an external database, Ignite provides distributing computations across all nodes in a balanced and fault-tolerant manner, Ignite automatically load balances jobs produced by a task as well as individual tasks by default it uses a round-robin algorithm, which distributes jobs sequential across the nodes.

When partition awareness is enabled the client is able to send the read request it has to the node that contains the correct data, when is turned off the client sends the request to NodeA and that node forwards the request to the node that

has the data (default mode), this only happens if Ingite is partitioned since the nodes don't have the same data

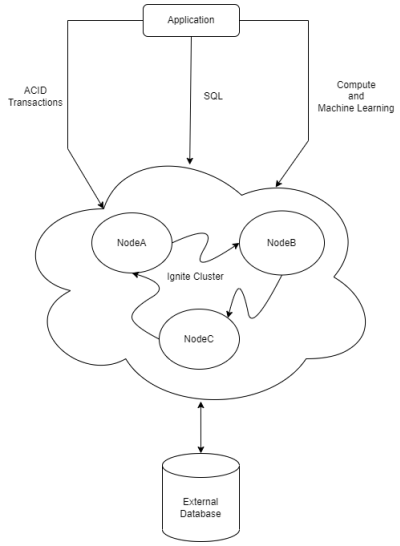

Fig. 3. Pipeline example

3 Experiment 1: Evaluating scalability using a single machine to deploy the ignite cluster

In this subchapter, we present the used experiment setup and describe the results obtained.

3.1 Experiment-Setup

The experiment was conducted on a single Google Cloud VM instance with specifications as listed in table 1. We decided to configure the cache to use the

Table 1. Google-Cloud VM specs

System	Google Compute Engine
OS	20.04.1-Ubuntu
CPU(S)	2 x Intel(R) Xeon(R) CPU @ 2.20GHz
RAM	16 GB

mode PARTITIONED as this is described to be the most scalable configuration [2].

To evaluate the scalability of an ignite cluster, we designed a simple experiment to determine the average throughput per number of cluster nodes. We deployed the ignite cluster using docker swarm and varied the number of running cluster nodes. For each number of nodes the same test was performed:

1. A client connects to the ignite cluster and creates a cache.
2. For a fixed period of time the client sends requests to the cluster that trigger cache operations. Each request is randomly assigned to either be a Cache-Put (write) or Cache-Get (read).
3. The client counts the number of cache operations it was able to initiate.
4. Once the time periods end has been reached, the client clears and deletes the cache.

We ran the experiment for 1,2 .., 20 cluster nodes. The time period set for the client to trigger cache operations was set to 10 minutes. Each cache operation wrote or read a random String of a fixed size (20 bytes) to or from the cache. Knowing the fixed time period, the fixed data size and the counted number of operations the throughput could be calculated.

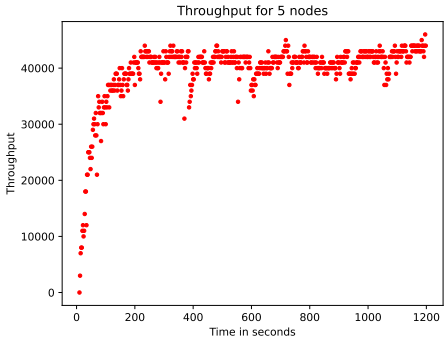

Fig. 4. Course of throughput values over 20 minutes.

During initial test runs we noticed that the throughput at the beginning of each measurement appeared to be significantly worse than when measured towards the end. Figure 4 shows a plot of the measured throughput values during a 20-minute run. It can be seen that the throughput rises sharply at the beginning before it settles at a stable level of ca. 40_000 bytes/second at approximately 300 seconds (5 minutes).

In order to maintain a measurement duration of 10 minutes, we decided to run the experiment for 15 minutes for each number of nodes, but only to evaluate

the measurements from the 5th minute onwards. The main results can be viewed in the file *./result/results_2022-10-14_23-35-22.csv*. All further results referred to in this report can be found in the same directory (and its subdirectory).

3.2 Results

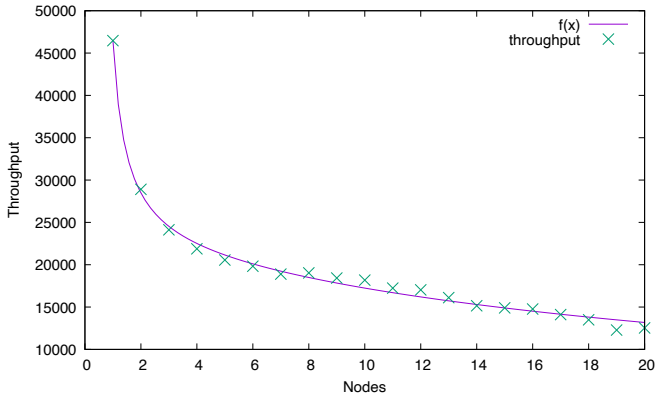

Fig. 5. Measured mean throughput and obtained model for the Universal Scalability Law.

Running the experiment as described in the previous subchapter and calculating the mean throughput per number of nodes (see figure 5) yielded following findings:

- Each increase in the number of cluster nodes lead to the throughput decreasing. This confirms to the official documentation [4] suggesting that the best performance is reached with running a single node per host machine.
- The proportional throughput loss per added node decreases with an increasing number of nodes. In particular going from a single node to two nodes resulted in a loss of ca. 38% which is significantly higher than for each following increase of nodes. This could be explained by the extra networking overhead imposed to be less impactful if the cluster already consists of many nodes.

Using the provided jar-file we also obtained the parameters for a model of the Universal Scalability Law.

$$X(N) = \frac{\lambda N}{1 + \sigma(N-1) + \kappa N(N-1)}$$

The base performance was calculated to be $\lambda = 46474,3471252964$, the serial part to be $\sigma = 2,1091123385$ and the crosstalk to be $\kappa = 0,0774972116$. As the

value of the serial part is higher than 1 (and therefor out of the possible range of 0 to 1) it is likely that the results cannot fully be explained relying solely on the Universal Scalability Law. To ensure that the results were not heavily influenced by either the CPU or the RAM running at maximum capacity we took snapshots of their utilisation (see files *./results/average_cpu* and *./results/average_mem*) which showed that even for a cluster of 20 nodes the CPU was running at less than 70% and the RAM at ca. 80% and we therefor assumed that this was the case. We therefor concluded that in order to better understand the measured scalability of the ignite system it is necessary to conduct further experiments.

4 Experiment 2: Evaluating the scalability of an ignite-cluster hosted accross multiple VMs

The previous experiment has raised questions regarding ignites scalability. More factors than just the number of nodes need to be tried and evaluated therefore we designed an experiment incorporating fractional factorial design.

4.1 Fractional Factorial Design

The Fractional Factorial Design measures only a specific combination of the factor levels. This needs to be carefully planned to best capture possible interactions. This allows for conducting an experiment with less effort than having to test each possible combination of factors, but leads to a higher probability of inaccuracies. The Fractional Factorial Design is particularly useful when it is known that some factors do not interact.

We evaluated following factors:

- Cache_Size: Size of the ignite cache used to store the key-value entries we write and read to the system.
- Nodes: The number of Ignite servers in the cluster that are each hosted on a separate VM.
- Cache_Mode: there are 2 cache modes. The cache is *partioned* when data is distributed over all server nodes. The cache is *replicated* when each node holds all data.
- Persistence: Can be enabled or disabled. When enabled Ignite stores all data on disk. Ignite stores each partition in a separate file on disk [5].
- CPU_Cores: The number of cores each host is assigned.
- Partition Awareness: If active thin clients are able to send requests directly to the node that contains the requested data.

As in fractional factorial design not all factors are trialed to the same extent we ordered the factors according to our expectations in regards to their impact on throughput and set up the sign table (table 2) accordingly. The results of our first experiment lead us to believe that the number of nodes may have a

significant impact. Partion Awarness and Cache Mode influence how a clients request is routed which makes us assume that these are also important factor to extensively test. Since Persistence is an important feature of Apache Ignite and we expect that turning it on or off would also impact highly throughput. We therefor consider these four factors as the main factors in our experiment design.

Experiment	**Nodes** **A**	**Partition Awareness** **B**	**Cache_Mode** **C**	**Persistence** **D**	**Cache_Size** **A*B**	**CPU_Cores** **C*D**
1	-1	-1	-1	-1	1	1
2	1	-1	-1	-1	-1	1
3	-1	1	-1	-1	-1	1
4	1	1	-1	-1	1	1
5	-1	-1	1	-1	1	-1
6	1	-1	1	-1	-1	-1
7	-1	1	1	-1	-1	-1
8	1	1	1	-1	1	-1
9	-1	-1	-1	1	1	-1
10	1	-1	-1	1	-1	-1
11	-1	1	-1	1	-1	-1
12	1	1	-1	1	1	-1
13	-1	-1	1	1	1	1
14	1	-1	1	1	-1	1
15	-1	1	1	1	-1	1
16	1	1	1	1	1	1

Table 2. Fractional Factorial Design

We confounded the left over factors as following: Cache_Size=Nodes*Partition Awareness and CPU_Cores=Cache_Mode*Persistence. We believe that these are appropriate choices as in both cases we expect that the factors are not inter-dependent with the main factors, making it easier to see the impact of the confounded factors.

4.2 Factor Levels

Factor	**-1**	**1**
Nodes	2	4
Partition Awareness	No	Yes
Cache_Mode	Replicated	Partioned
Persistence	Active	Inactive
Cache_Size	512	2048
CPU-Cores	1 (2 vCPU)	2 (4 vCPU)

Table 3. High and Low levels assigned for selected system factors.

When deciding on levels for the system factors some restrictions imposed by Google Cloud had to be considered. Primarily we had to limit the number of vCPUs distributed over server nodes and the client to a total of 24. This number of available vCPUs for the Ignite server cluster was further reduced by the need to assign 8 vCPUs to the machine hosting the clients. This was necessary to ensure that the throughput measured in the experiment was not restricted by the Clients-Host being overloaded. As we expected the factor CPU-Cores to have little impact on the systems performance we set the levels to the smallest possible combination (1 core vs 2 cores). Furthermore as Ignites discovery mechanism is not compatible with Docker Swarm we decided to have each Ignite node run directly on the VM (rather than inside a container) and discover each other using the internal VM IP addresses. With the maximum number of available vCPUs being 16 the maximum we set the high level for Nodes to 4. For Cache_Size we opted for the lowest advised size of 512 Mb as the low level and and the fourfold as the high level. Here we aimed to both push the system to its limit (in terms of cache size) with the low level and to ensure that the hosts RAM is not overloaded when applying the high level. All left over factors are binary modes were we set the high and low level according to our expectations of which mode would yield the higher performance.

4.3 Workload

The experiment was conducted with a real case use of Ignite in mind of Raiffeisen Bank [7], in this case they use 2 types of workload OLTP (Online Transaction Processing) and OLAP (Online Analytical Processing). While OLTP is used as a transactional app with many database tables and more write, OLAP is used for analytics and reporting having few database tables and more reads. For our workload we opted for something similar to OLAP, because we were more interested on testing Ignite limitations on this type of workload, where our focus is going into reads with client having a 85% of going for a read and the rest of going for a write.

After the cache was was warmed up for five minutes (detailed in subchapter 3.1) the workload is put on the system by having clients attempt to perform as many operations (using the ratio mentioned above) as possible over a fixed amount of time (10 minutes). Operations are performed sequential, meaning the client waits for the response for operation 1 before iniating operation 2.

The workload is scaled according to the number of Ignite nodes present in the cluster, meaning for every node in the cluster there exists a client putting workload onto the cluster.

Clients access the cluster by connecting to a random node from the cluster. If Partion awarness is set to *Inactive* the client will keep sending its request to that node. If the node cannot serve the request on its own it forwards the request to other nodes in the cluster, and once it receives a response it passes it on to the client. In case Partion awarness is *Active* the client has insight on which node holds the partion needed to answer is his request, and is therefor able to address the correct node directly.

4.4 Processing results

The results were obtained by each client writing to a shared result file. Throughput, which serves as the main metric we use to analyze Ignite, is calculated using the same idea as in the previous experiment (subchapter 3.1), meaning we count the number of operations (each having a payload of the same size) performed over a fixed amount of time and use that as a basis for calculation. Next to the throughput, we also measure the time write-operations and read-operations took. The raw experiment results may be viewed in the file *results/results.csv*. A simple python script (results/summerize_results.py) may be used to improve readability for the results (see *summed_up.csv*).

The analysis was done by porting the results to excel spreadsheets that can be viewed in the directory *results/spreadsheets*.

4.5 Evaluating the results

Our results show that the measured throughput reached from a minimum of 27169,25 bytes/sec to a maximum of 56756,50 bytes/sec, which is more than double. Table 4 show the level combinations that yielded the worst and best throughput.

Factor	**Level for min throughput**	**Level for max throughput**
Nodes	4	4
Partion Awareness	Inactive	Active
Cache Mode	Partioned	Partioned
Persistence	Active	Active
Cache Size	512 Mb	512 Mb
Cpu Cores	4	4

Table 4. Worst and best configuration in respect to measured throughput.

The estimated impact of the trialed factors (rounded) are presented in table 5.

The results shot that from the trialed factors the number of nodes, persistence mode, Cache Size and CPU_Cores have little to no impact on Ignite overall performance measured by throughput. As the the number of nodes has almost no impact on the time read-operations take (0%), in contrast to an estimated impact of 9% for write operations, we concluded that for our workload (focused on reads) the number of nodes is irrelevant. We further suspect that the restrictions we had to consider when defining the high level for Nodes (only 4) did not allow

Factor	**Effect**	**Sign**	**Impact Throughput**	**Impact Reads**	**Impact Writes**
Nodes	Negative	A	1%	0%	9%
Partition_Awarness	Positive	B	20%	17%	18%
Cache_Mode	Negative	C	22%	43%	19%
Persistence	Positive	D	0%	11%	0%
Cache_Size	Positive	A*B	3%	4%	4%
CPU_Cores	Positive	C*D	5%	2%	2%
Partion_Awarness & Cache_Mode	Positive	B*C	26%	18%	24%
Not further inspected			23%	23%	24%

Table 5. Estimated impact of factors trialed.

us to adequately estimate the actual impact, as factor like cross talk may only have a significant impact when dealing with a greatly larger number of nodes.

For the factors persistence (on/off), CPU_Cores we suspect that the workload we chose to put on the system was not stressing each part enough for there to be a significant drop in performance. I.e. a workload that does not focus on reading and writing as fast as possible (as our does) but instead on performing computationally expensive transactions might show that persistence could be of significant impact. Cache_Size appears to have little impact for both reads and writes and therefor no significant impact on the overall throughput.

As for significant factors Partion Awarness and Cache Mode and especially their interplay appear to be the most relevant factors (out of the ones we evaluated) for our setup, all having impacts greater than 20% for throughput. Here activating Partion Awarness increased performance and changing Cache_Mode from replicated to partioned decreased performance. This can be explained by considering that our workload focuses on read operations and that these factors define the number of nodes a read-request traverses on average. For all combinations of levels that are **not**:

- cache is partioned and
- client is not partion aware

Clients always receive the requested data from the first node they contact. This is due to that either the first node holds the data because its replicated or the client knows what node to contact (is partionaware). This is different for the left over case where, the read-request can only be served by traversing just one node, if that node holds the requested data on chance (as data is distributed over all nodes). The chance for this to be case, assuming the data is perfectly/equally distributed across all nodes, can be assumed to be $1/nodes$. Nodes are partion aware, meaning they know which node holds the data, hence a request is server by traversing a maximum of two nodes. Putting that together the probability of a read-request needing to traverse two nodes can be assumed to be $X =$

$1 - (1/nodes)$. For only 4 nodes the probability of two nodes being necessary to serve a read-request is already at 80%. Consequently 80% of the send requests (in that setting) lead to a case where we suspect that idle time, where the first node is waiting for the second nodes response, and cross talk lead to significant drop in throughput.

Based on our findings we argue that when deploying a Ignite cluster one should avoid the worst case configuration of clients not being partion aware and Cache not being replicated. Ignite offers the best performance when clients are partion aware and the cache is partioned.

4.6 Impact of the factor network

In our experiment we did not directly evaluate if the network could be a potential bottleneck. However during trial runs of our experiment we unintentionally used a configuration where the client was not always hosted on a machine sharing the same region as the machines hosting the ignite cluster. The raw results of that not completed attempt can be viewed in *results/archive/results_Network.csv*. Here it became obvious that once the client did was not hosted on the same region as the cluster anymore the throughput dropped from a range of ca. 30000 to 55000 to a range of 796 to 810, implying a perfomance loss of at least 97%. Considering that our experiment only saw a perfomance drop (best vs worst configuration) of ca 50% this leads us to believe that the most important limiting factor of Ignite is likely the network.

5 Conclusion

Based on the first experiment we reached the following conclusions. First, Apache Ignite does not seem to scale well if the cluster is hosted on a single machines as shown in figure 5. We also identified that the cache needs around 5 minutes of warm up, which we incorporated when configuring our experiment setups. As our findings were unexpected we concluded, that further experiments i.e. varying the number of host machines appear to be necessary to better evaluate Apache Ignites scalability, which serves as motivation for the second experiment we conducted.

The results of the second experiment lead us to conclude, that Cache_Mode and Partition Awareness appear to be the most impactful factors. We advise to run Ignite with the cache being partioned and the clients having knowledge of how these partions are distributed over the cluster, hence the client being partion aware. Finally we discussed that while the configuration of our trialed factors may lead to perfomance loss of 50% (or performance gain of 100%) the perfomance loss when the client is operating from a different region than the ignite cluster (more than 97%) implies that the network is most likely the main bottleneck of Ignite.

References

1. Apache Ignite: Data Partitioning. https://ignite.apache.org/docs/latest/data-modeling/data-partitioning#rebalancing (2015), online; accessed 3 November 2022
2. Apache Ignite: Data Partitioning. https://ignite.apache.org/docs/latest/data-modeling/data-partitioning (2015), online; accessed 14 October 2022
3. Apache Ignite: Distributed In-Memory Cache. https://ignite.apache.org/use-cases/in-memory-cache.html (2015), online; accessed 14 October 2022
4. Apache Ignite: Generic Performance Tips. https://ignite.apache.org/docs/latest/perf-and-troubleshooting/general-perf-tips (2015), online; accessed 13 October 2022
5. Apache Ignite: Ignite Persistence. https://ignite.apache.org/docs/latest/persistence/native-persistence (2015), online; accessed 3 November 2022
6. Apache Ignite: What is Apache Ignite? https://apacheignite.readme.io/docs (2020), online; accessed 12 October 2022
7. GridGain Systems: Apache Ignite, Load Reduction and System Scaling for Banking. https://www.youtube.com/watch?v=Mhtt2QL_qCQ (2021), online; accessed 3 November 2022
8. Shamim Ahmed Bhuiyan, Michael Zheludkov and Timur Isachenko: High Performance in-memory computing with Apache Ignite. http://leanpub.com/ignite (2018), online; accessed 13 October 2022