ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-TUBL01

DISTRIBUTED CACHING AT CLOUD SCALE WITH APACHE IGNITE FOR THE C2MON FRAMEWORK

T. Oliveira*, D. Martin Anido, M. Braeger, B. Copy, S Halastra, A. Papageorgiou Koufidis
CERN, Geneva, Switzerland

Abstract

The CERN Control and Monitoring platform (C2MON) [1] is an open-source platform for industrial controls data acquisition, monitoring, control and data publishing. Its high availability, fault tolerance and redundancy make it a perfect fit to handle the complex and critical systems present at CERN. C2MON must cope with the ever-increasing flows of data produced by the CERN technical infrastructure, such as cooling and ventilation or electrical distribution alarms, while maintaining integrity and availability. Distributed caching [2] is a common technique to dramatically increase the availability and fault tolerance of redundant systems. For C2MON we have replaced the existing legacy Terracotta [3] caching framework with Apache Ignite [4]. Ignite is an enterprise grade, distributed caching platform, with advanced cloud-native capabilities. It enables C2MON to handle high volumes of data with full transaction [5] support and makes C2MON ready to run in the cloud. This article first explains the challenges we met when integrating Apache Ignite into the C2MON framework, and then demonstrates how Ignite enhances the capabilities of a monitor and control system in an industrial controls environment.

INTRODUCTION TO C2MON

C2MON is an open-source monitoring platform developed at CERN. C2MON acts as the backbone of the Technical Infrastructure Monitoring system (TIM) that is used to monitor and control CERN's technical services from the CERN Control Centre (CCC) [6]. The main function of TIM is to provide reliable and real-time data to CCC operators about the state of CERN's widely distributed technical infrastructure. To handle such a large volume of information while maintaining data integrity, C2MON uses Java Message Service (JMS) [7] technologies together with caching technologies. Caching involves storing information in a separate low-latency data-structure for a period of time to be reused and consequently minimizing the cost of re-accessing it [2]. The existing C2MON caching layer relied on a legacy Terracotta Ehcache framework [3]. Ehcache is a widely-used Java-based cache that is fast, lightweight and can be scalable through the use of a Terracotta Server that provides distributed caching capabilities [3].

C2MON uses a 3-tier architecture, as presented in Fig. 1, that composes a Data Acquisition (DAQ) Layer, the Server Layer and a Client Layer. The DAQ layer is responsible for acquiring data from specific sources and publishing it to the C2MON server tier. The Client layer provides various service classes that allow to interact with the server. The Server layer, which is the core part of C2MON, is responsible for receiving and handling the data.

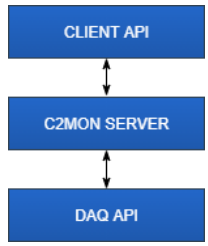

Figure 1: C2MON Architectural Overview.

Most of the information in C2MON is stored and used in the form of Tag which changes frequently as new data from the DAQ is received and evaluated by the system. In order to evaluate the correct and normal values of each Tag, C2MON provides an Alarm mechanism. The Alarm is a declaration associated with a Tag and contains a condition specifying the legal values of that Tag. If the new value received is outside the legal value range, the Alarm is activated and pushed to the client. Finally, the C2MON server layer also provides a rule engine with a set of operations, that allows expressing complex computations, comparisons and conditions.

INTRODUCTION TO APACHE IGNITE

"Apache Ignite is an open-source memory-centric distributed database, caching and computing platform" [4].

Ignite provides a simple interface to work with large data sets in real time. It is written in pure Java, based on Spring and supports different technologies like Java, C# and C++. The main capabilities that Apache Ignite provides are

- Elasticity: The Ignite cluster can grow horizontally simply by adding new nodes over a TCP connection.
- Persistence: Cache entries can be persisted on a file system or in an RDBMS (Relational Database Management System).
- Distributed computing: Apache Ignite provides a set of APIs that facilitate the distribution of computation and data processing across the nodes in the cluster for better performance; it simplifies greatly the development of a microservice-based architecture.
- Streaming: Ignite allows the processing of continuous streams of data (which C2MON uses to receive cache events asynchronously through continuous queries).

Ignite includes the notion of client and server nodes, where a node is a single Ignite instance running in a JVM (Java Virtual Machine). In a client server architecture both client and server nodes are interconnected with each other. The server, which can be constituted by a single node or a group

* tiago.marques.oliveira@cern.ch

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-TUBL01

of nodes forming a cluster, handles all the data storing processes and computing of data. The client node is usually embedded with the application code and acts as an interface to run operations in the cache like inserting or retrieving data, but can in some cases also participate in computing tasks [4]. Apache Ignite comprises a memory-centric architecture that can be used as an In-memory or distributed database. This memory-centric storage allows to store data and indexes both in-memory and in-disk with the same data structure and enables executing SQL queries in both with optimal performance.

HOW CACHING WORKS IN C2MON

In such a fast paced environment as the one present at CERN where it is crucial to maintain the correct functionality of critical systems, it is essential to have a reliable monitoring platform capable of providing high availability and fault tolerance when dealing with the increasing flows of data produced by the CERN Technical Infrastructure. In order to achieve those requirements, C2MON relies on caching technologies. The cache in C2MON acts as a working memory that allows for stored information to be retrieved as fast as possible. The database is a persistent storage, and serves as a backup solution, that is responsible for storing long term data and to populate the cache when necessary.

C2MON uses a classic client/server cache with a distributed cluster. This caching layer serves as a middleware that reduces time-consuming operations of fetching data from the database by storing the objects that are most used. When the cache starts up, it must be warmed up : cache loaders in C2MON are responsible for converting the data loaded from the database to POJOs (Plain Old Java Objects) and storing them in the cache [8].

The infrastructure of C2MON heavily relies on the use of ActiveMQ [9] to transport data. ActiveMQ is an open source, Java Messaging Service (JMS) compliant, message-oriented middleware supported by the Apache Software Foundation. The number of messages that are received can reach up to a few millions per day, and C2MON has to process all of them in real-time while maintaining data consistency. This processing involves the evaluation of Tag values, activation or termination of Alarms, rule calculation and event propagation. The calculation of rules in C2MON is a very expensive process that involves many successive calculations and can even be recursive. As the rate is so fast, some sort of rapid access memory is necessary to store intermediate values. Since JMS is only responsible of receiving and passing data and does not allow the storage of data, a new component is needed to fulfill this requirement. This logic could also be implemented directly in the database layer, as C2MON uses a relational database like Oracle. But this would require business logic to be implemented outside the C2MON server core, it would require the use of Oracle-specific Procedural Language and would lead to a lot of latency when persisting all the intermediary values. The solution is the use of a distributed working memory like Apache Ignite. This helps to achieve availability and consistency of data.

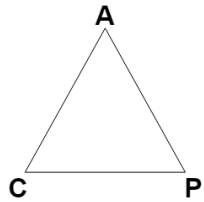

Figure 2: CAP theorem.

ADVANTAGES OF APACHE IGNITE

The Apache Ignite architecture is flexible enough to be used in various architecture patterns and styles. One possible approach that facilitates local and integration tests is to run Ignite embedded with the application where the node runs on the same JVM with the application. In a production environment a better approach is to have the Apache Ignite server nodes in a separate JVM and client nodes that connect remotely to the servers. The great advantage of this approach is the increasing flexibility that allows Ignite servers to be taken down and restarted without harming the cluster or the correct functioning of the application they support [10]. Ignite also offers the possibility of using three approaches of caching topology, where each cache mode is individually configured for each cache. In C2MON, Apache Ignite was configured with a replicated caching topology in order to achieve the highest possible level of performance. In this approach, cached data is replicated to all members of the cluster so it is available immediate use without waiting and is thus providing the fastest read-access of all caching topologies. The main downside of this approach is that writing operations are very expensive since updating a replicated cache requires pushing the new version to all cluster nodes which may limit the scalability with high frequency of updates [10].

CAP Theorem

There is a widely known idea that needs to be taken into account when dealing with distributed computing theory, which is the CAP Theorem introduced by Eric Brewer [11]. This theorem presents the fundamental trade-off between consistency, availability and partition tolerance. Figure 2 presents the graphical representation of the CAP Theorem where the three properties represent:

- Consistency that implies that all the nodes in the cluster have the same data.
- Availability of the system that should always answer the queries if possible.
- Partition Tolerance meaning that if a break in the communication occurs, the system will continue to work as expected.

Based on this theorem, Apache Ignite can be classified as a CP system, meaning that availability is sacrificed for consistency of data and partition-tolerance, since it is ACID (Atomicity, Consistency, Isolation, Durability) compliant,

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-TUBL01

supporting distributed transactions with partitioned tolerance [10]. Ignite can also be considered as an AP system, since it offers two types of transactional modes for cache operations: atomic or transactional. In atomic mode Ignite supports multiple atomic operations, successively, where each DML (Data Manipulation Language) operation will either succeed or fail without any data being locked, giving it a high performance. On the other hand, in transactional mode, the DML operations can be grouped in one transaction and the data will be locked [10]. This second approach is used by Ignite in C2MON, with CP prioritised, where the consistency of data is crucial.

DIFFICULTIES AND CHALLENGES FACED

Having covered the background to this project and giving an overview of the caching system, the focus now shifts to the difficulties and challenges faced during the process of replacing the caching system.

Code Rigidity

One major problem with the design of C2MON was the rigidity of its caching layer design. Rigidity is the tendency for software to be difficult to change and where every change can cause subsequent changes in dependent modules [12]. This was due to the tight coupling of C2MON to the current caching technology, so the change would involve refactoring a major part of the current architecture.

Testing

It is crucial to have a good testing environment to ensure that the software is working as expected, and to detect and fix any eventual errors or bugs, when dealing with large and critical systems like C2MON. This should include not only unit tests to validate individual components but also integration tests to check the good behavior of those units interacting together. Integration testing environments should mimic production as much as possible, so that potential issues in production can be reproduced and resolved with certainty. This step becomes even more difficult with the increasing number of external dependencies like ActiveMQ, Oracle database and Apache Ignite, in the case of C2MON.

Cloud Readiness

Over the past years, CERN has embraced cloud technology which presents significant advantages. It allows a more agile sharing of resources and it simplifies the reusing and duplication of entire groups of machines for testing purposes. Cloud-based deployment file systems are also usually transient and network interfaces are typically allocated on the fly with a randomly-generated hardware address and attached to a local, private and non-routable network. Furthermore, the life cycle of a cloud container hosting an application is linked directly to its main process, which must consequently be managed at level of the hosting cloud, as opposed to the host they are running on [13]. C2MON was initially designed around 2009, when cloud technologies were not as ubiquitous as today, and its legacy architecture makes it impossible to take advantage of all the capabilities of a platform like Apache Ignite that is considered "cloud-ready". An example of this is the challenge node discovery in a cloud. Nodes in a cloud that wish to regroup in a cluster must usually employ broadcast messages, but this is disallowed at CERN for performance and security reasons. As a result, C2MON must rely on a fixed list of IP addresses, severely limiting how fast it can be deployed and reconfigured over new hardware or a dynamic cloud environment.

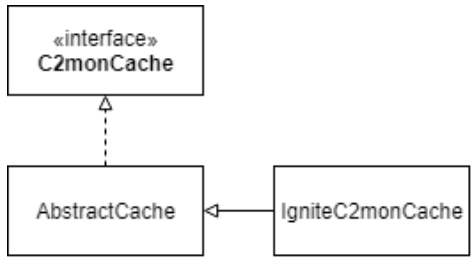

Figure 3: C2MON Cache Abstraction Class Diagram.

SOLUTIONS

The remainder of the paper will describe the redesign of the C2MON platform caching layer, and how the design choices have helped to overcome these aforementioned challenges and achieve the goal of integrating Apache Ignite into the C2MON framework.

Code Refactoring

In order to change the caching layer technology of C2MON, it was necessary to make a big code refactoring of various modules that were tightly coupled to it. To minimize the rigidity of C2MON's infrastructure, the caching layer was rebuilt depending completely on abstractions, decoupling it from the technology in place. Every dependency in the design should target an interface, or an abstract class, and never a concrete class that is much more prone to changes [12]. This approach has the disadvantage of losing some of the capabilities of Apache Ignite, but has the advantage of making it easier to change the caching technology in the future (if necessary), without major code changes. Figure 3 presents a snippet of the class diagram that corresponds to the new C2MON cache abstraction layer.

The `C2monCache` interface is the base of the entire C2MON caching layer, and contains all the necessary methods that interact with the cache, like inserting elements in the cache and retrieving them and querying elements in the cache, that was purposely implemented in a very simple way so that it could be implemented by any future alternative caching technology. This interface is implemented by an abstract class, `AbstractCache`, that hides the underlying implementation of the caching technology, and is also responsible for managing the caching update flow, the cache loaders and the cache listeners. The cache loaders are used to warm up the cache on startup and the cache listeners are used

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-TUBL01

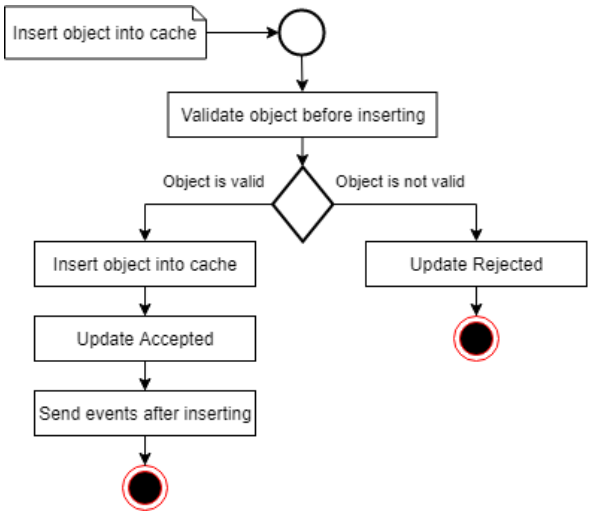

Figure 4: C2MON Cache Insertion flow.

to propagate all caching events. The `IgniteC2monCache` is the concrete class that contains the Apache Ignite implementation of `C2monCache`. Another implementation choice was the use of Java Generics throughout the whole caching implementation. Generics allows to abstract over types and the core idea is to express intent, and mark the respective objects as being restricted to contain a particular data type [14]. The `Cacheable` interface is implemented by all objects that reside in C2MON caches, creating another layer of abstraction and removing the possibility of run time errors during execution. Although Apache Ignite supports ACID transactions this might not be present in other caching technologies, and data consistency must be guaranteed for the good functioning of C2MON. Figure 4 displays an activity diagram of how the insertion in the cache is processed to respect cache consistency. When a new object is inserted in the cache it goes through a validation process first that guarantees that the object is complete and correct, and it is not older than the previous update, otherwise the update operation is discarded. In case the validation passes, the object is successfully inserted in the cache and corresponding events are propagated to any potential cache listeners.

Testing

The lack of a proper testing environment, made it difficult to fully test the correct integration of C2MON with its various external dependencies. The alternative used to overcome this problem was the use of Testcontainers [15]. Testcontainers is a Java library that supports JUnit tests and that can execute those tests against instances of any software that can run in a Docker container. To use Testcontainers, all of C2MON components had to be ”containerized” for the Docker container platform. Testcontainers encapsulate the management of the Docker images and spins up the required containers during the tests and tears them down once the test execution is finished. It provides a faster feedback loop for integration tests, as all the containers start with a clean and known state which increases the reliability of the tests.

CONCLUSION

The usage of Apache Ignite has proven to be a robust and capable solution, that enhanced the capabilities of the C2MON framework. However, some of its capabilities could not be completely exploited at CERN because of the way C2MON is deployed. Although the main goal was to change the caching technology of C2MON and to integrate Apache Ignite, the opportunity was also taken to sanitize and refactor the entire caching layer. This was done following the best practices for software development and testing, employing newer features like the implementation of generics and the use of Java streams. Product testability proved to also be a very important factor that had to be improved to ensure that C2MON behaved the same way as before the Ignite integration : Docker technology (through Testcontainers) proved an effective and simple way to perform extensive, repeatable integration testing of the C2MON platform.

REFERENCES

[1] M. Braeger *et al.*, “A customizable platform for high-availability monitoring, control and data distribution at CERN”, in *Proc. ICALEPCS 2011*, Grenoble, France, 2013, pp. 418–421.

[2] Greg Luck and Brian Oliver, “Java™ Caching API - The Java Caching API is an API for interacting with caching systems from Java programs”, Dec. 2013. `https://download.oracle.com/otn-pub/jcp/jcache-1_0-fr-spec/JSR107FinalSpecification.pdf`

[3] Software AG, “About Terracotta Ehcache”. Apr. 2019. `http://documentation.softwareag.com/onlinehelp/Rohan/tc-ehcache_10-3/10-3_About_Ehcache.pdf`

[4] Shamim Bhuiyan and Michael Zheludkov, “The Apache Ignite Book - the next phase of the distributed systems”, 2019.

[5] Paul Parkinson, “Java™ Transaction API (JTA) - Version 1.2”, May 2013.

[6] J. Stowisek, A. Suwalska, and T. Riesco, “Technical infrastructure monitoring at CERN”, in *Proc. EPAC’06*, Edinburgh, Scotland, 2006, paper TUPLS135, pp. 1822-1824.

[7] Richard Monson-Haefel and D. Chappell, “Java™ Message Service (JMS)”, 2000.

[8] Szymon Halastra, “Refactoring of the CERN in-memory Data grid”, 2019.

[9] Bruce Snyder, Dejan Bosanac, and Rob Davies, “Introduction to Apache ActiveMQ”, August 2008.

[10] Shamim Bhuiyan, Michael Zheludkov, and Timur Isachenko, “High Performance In-Memory Computing with Apache Ignite - building low latency, near real-time application”, 2017.

[11] Seth Gilbert and Nancy A. Lynch, “Perspectives on the CAP Theorem”.

[12] R. C. Martin, “Design Principles and Design Patterns”, 2000.

[13] B. Copy, M. Braeger, E. Mandilara, F. Ehm, and A. Lossent, “C2MON Scada Deployment on Cern Cloud Infraestructure”, 2017.

[14] Gilad Bracha “Generics in the Java Programming Language”, January 2004.

[15] Testcontainers, `https://www.testcontainers.org/`.