ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 6, June 2014

Infinispan as Key / Value data store

Sarith Divakar M[1], Ashina Tholiyil[2]

Lecturer, Computer Engineering, College of Applied Science, Kozhikode, India [1]

Lecturer, Computer Science, College of Applied Science, Kozhikode, India [2]

Abstract: Biggest challenges long-faced by todays applications are unit growth of knowledge. Quantity of knowledge created and consumed is growing in size. It will increase further when complexity of application increases. Centralized solutions for storing and retrieving data aren't possible for several reasons. First reason is that the inability to scale as and once required. Single purpose of failure hinders handiness of the applications and therefore reducing potency. Owing to these shortcomings of centralized systems we move towards data grids. Data grids act as a middleware which will store massive set of data across distributed applications. Data grids distribute large sets of data across a group of servers over a network, therefore forming a cluster. Middleware ought to be ready to serve numerous platforms and storage systems. Data grids have to be compelled to be optimized to scale go in environments that manufacture and use large amounts of data. Infinispan is that the resolution that has of these options.

Keywords: Infinispan, data grid, data structure, NoSQL

I. INTRODUCTION

Infinispan is a key/value NoSQL data store. It is written in Java. Infinispan data store can run in distributed mode, thus providing extreme scalability and high availability. Infinispan project aims to provide a data structure that is highly concurrent that make the most of modern multi-processor/multi-core architectures. At the same time it provides distributed cache capabilities. It exposes a JSR-107 compatible Cache interface (which successively extends java.util.Map) within which you will be able to store objects. Infinispan can run in native mode, its real price is in distributed mode wherever caches cluster along and expose an oversized memory heap. Distributed mode is a lot of powerful than straightforward replication since every information entry is displayed solely to a set range of replicas therefore providing resilience to server failures also as measurability since the work done to store every entry is constant in relevancy to a cluster size.

Infinispan provides huge heap and high handiness - If you've got one hundred blade servers, and every node has 2GB of house to dedicate to a replicated cache, you finish up with two GB of total information. Each server is simply a duplicate. On the opposite hand, with a distributed grid you wish one copy per information item - you get a one hundred GB memory backed virtual heap that's expeditiously accessible from anyplace within the grid. If a server fails, the grid merely creates new copies of the lost information, and puts them on alternative servers. Applications longing for final performance aren't any longer forced to delegate the bulk of their information lookups to an outsized single information server - the bottleneck that exists in over 80% of enterprise applications.

Infinispan offers extreme scalability options. Since data is equally distributed there's basically no major limit to the scale of the grid, except cluster communication on the network that is minimized to merely discovery of latest nodes. All data access patterns use peer-to-peer communication wherever nodes directly speak to every different, that scale alright. Infinispan doesn't need entire infrastructure closedown to permit scaling up or down. Merely add/remove machines to your cluster while not acquisition any down-time.

Data distribution is handled well in Infinispan data grids. Infinispan uses consistent hash algorithmic rule to see wherever keys ought to be situated within the cluster. Consistent hashing permits for reasonable, quick and in particular, settled location of keys with no want for more data or network track. The goal of data distribution is to keep up enough copies of state within the cluster thus it may be sturdy and fault tolerant, however not too several copies to forestall Infinispan from being ascendable.

Infinispan exposes a cache store interface, and several other superior implementations, together with JDBC cache stores, Amazon S3 cache stores, le system based cache stores, etc. Infinispan offers support for the favoured memcached protocol - with existing clients for nearly each popular language - in addition as associate optimized. Infinispan -specific protocol referred to as HotRod. This suggests that Infinispan isn't simply helpful to Java. Any major web site or application that wishes to require advantage of a quick information grid is going to be ready to do therefore.

II. INTERACTING WITH INFINISPAN

Infinispan is an open source data grid platform. Data grids are unremarkably used as low-latency, highly-available and elastic knowledge storage backend, usually as NoSQL solutions. Data grids are usually employed in addition to traditional databases, as a distributed cache for quick data access. There are two ways that during which you will be able to interact with Infinispan. One is in embedded mode, wherever you start Infinispan in stance inside your JVM. The opposite is client/server mode, wherever you start a remote Infinispan instance and hook up with it employing

ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 6, June 2014

a client connector. Your selection on that mode of interaction to use can depend upon variety of things, together with whether or not you are using Infinispan as a clustering toolkit to cluster your own framework, whether or not you propose to use Infinispan to cache database lookups, or whether or not you propose to interact with Infinispan from a non-JVM environment.

III. INFINISPAN CONFIGURATIONS

We will configure data grid in each full and partial replication of data, in each synchronous and asynchronous manner. In a fully replicated mode, all nodes in a cluster hold copies of all entries (if an entry exists on one node, it will also exist on all other nodes).

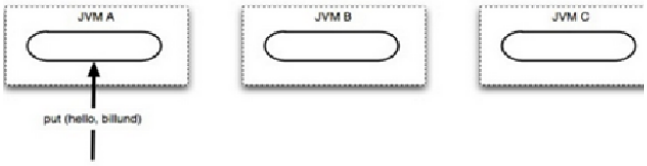

Figure 2.1: Cache running on top of JVM

Figure 2.1, shows Infinispan caches running on top of Java Virtual Machine (JVM). Data is inserted to cache running on top of JVM A and Figure 2.2 shows replication of data corresponding to keyword to all other caches.

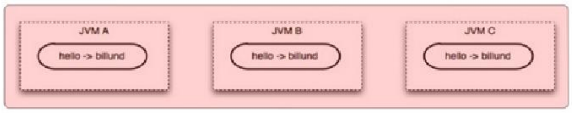

Figure 2.2: Full replication mode

In a part replicated mode, a fixed number of copies are maintained to supply redundancy and fault tolerance, no matter cluster size. Figure 2.3 shows replication of knowledge to JVM A and JVM B. This can be usually so much fewer than the amount of nodes within the cluster. A partially replicated knowledge grid provides a so much bigger degree of scalability than a completely replicated one. It's so the recommended clustering mode in Infinispan. Invalidation is a clustered mode that doesn't truly share any knowledge in the slightest degree, however merely aims to get rid of data which will be stale from remote caches.

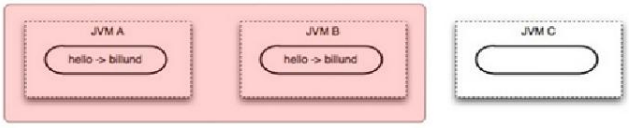

Figure 2.3: Partial replication mode

IV. INFINISPAN MODES

Traditionally, clients have interacted with Infinispan in an exceedingly peer-to-peer (P2P) fashion where Infinispan and therefore the client code that accessed it lived within same VM. Once Infinispan is queried during this method, it's thought of to be accessed in an embedded fashion, as shown in the figure 2.4.

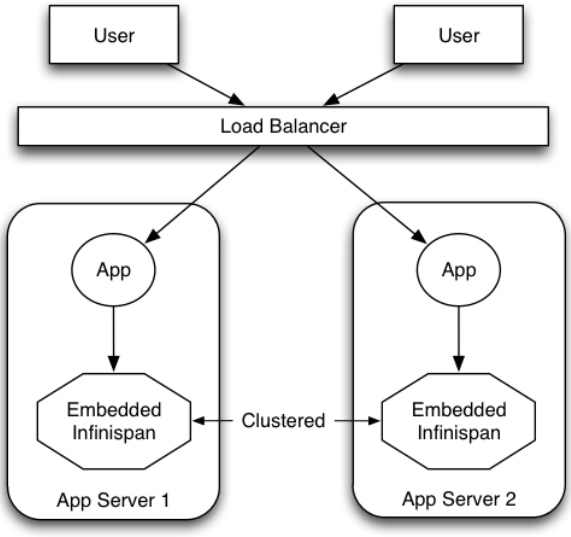

Figure 2.4: Peer to peer access

However, there are situations when accessing Infinispan during a client-server mode may build a lot of sense than accessing it via P2P. For example, attempting to access Infinispan from non-JVM surroundings. Since Infinispan is written in Java, if somebody had a C++ application that needed to access it, it could not access in a P2P manner. On the opposite hand, client-server would be absolutely suited here assumptive that a language neutral protocol was used and also the corresponding client and server implementations were obtainable. Figure 2.5 shows client-server mode

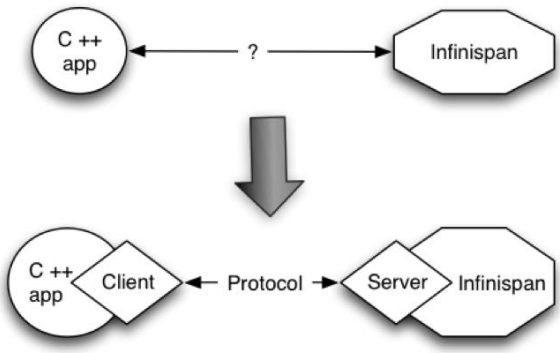

Figure 2.5: Client-server mode

Infinispan users wish to own associate elastic application tier wherever you start/stop business process servers terribly frequently. Now, if users deployed Infinispan organized with distribution or state transfer, start-up time may well be greatly influenced by the shuffling around of information that happens in these things. Thus within the figure 2.6, assuming Infinispan was deployed in P2P mode, the app within the second server couldn't access Infinispan till state transfer had completed.

ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 6, June 2014

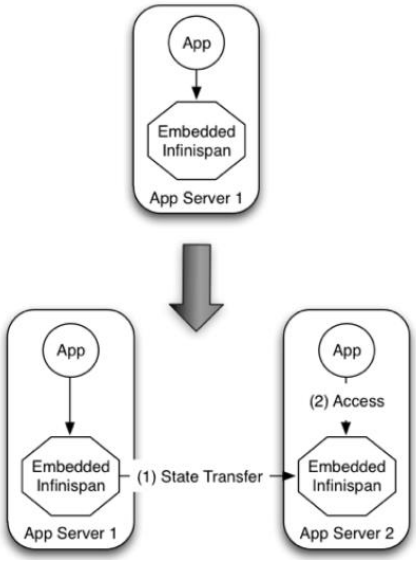

Figure 2.7: Elastic issue with P2P

This effectively implies that remarking new application-tier servers is compact by things like state transfer as a result of applications cannot access Infinispan till these processes have finished and if the state being shifted around is giant, this might take your time. This is often undesirable in associate degree elastic surroundings wherever you would like fast application-tier server turnaround and predictable start-up times. Issues like this will be resolved by accessing Infinispan during a client-server mode as a result of beginning a brand new application-tier server is simply a matter of beginning a light-weight client that may connect with the backing data grid server. No need for rehashing or state transfer to occur and as a result server start-up times may be additional predictable that is extremely necessary for contemporary cloud-based deployments wherever physical property in your application tier is vital. Figure 2.7 shows achieving elasticity through client server mode.

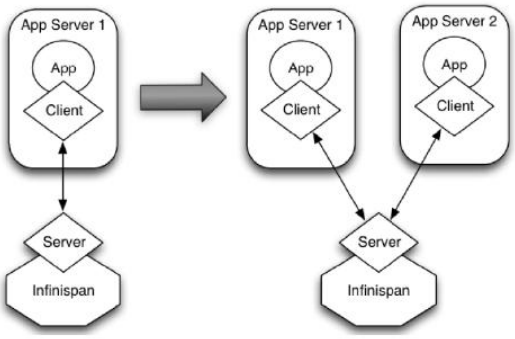

Figure 2.7: Achieving elasticity

Other times, it's normal to seek out multiple applications needing access to data storage. In these cases, you may in theory deploy an Infinispan instance per each of these applications however this might be wasteful and troublesome to keep up. In case of databases you do not deploy a database in your applications. So, instead you may deploy Infinispan in client-server mode keeping a pool of Infinispan data grid nodes acting as a shared storage tier for your applications as shown in figure 2.8.

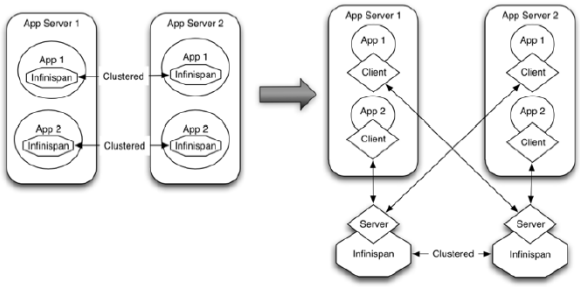

Figure 2.8: Shared data storage

Deploying Infinispan this manner conjointly permits you to manage every tier independently, for instance, you'll upgrade you application or app server without bringing down your Infinispan knowledge grid nodes. Client-server Infinispan still has disadvantages over P2P. Firstly, P2P deployments are easier than client-server ones as a result of in P2P, all peers are equals to every alternative and thus this simplifies deployment. Client-server Infinispan requests are probably to require longer compared to P2P requests, attributable to the serialization and network cost in remote calls. So, this is often a crucial issue to require in account once planning application. As an example, with replicated Infinispan caches, it would be higher to own light-weight communications protocol clients connecting to a server side application that accesses Infinispan in P2P mode, instead of having a lot of heavyweight client side apps talking to Infinispan in client-server mode, notably if data size handled is very massive. With distributed caches, the distinction may not be therefore huge as a result of even in P2P deployments, you are not sure to have all data accessible locally. Environments wherever application tier elasticity isn't therefore necessary, or wherever server side applications access state-transfer-disabled, replicated Infinispan cache instances are amongst eventualities wherever Infinispan P2P deployments are often additional suited than client-server ones. All Infinispan server modules are supported identical pattern wherever the server backend creates an embedded Infinispan instance and if you begin multiple backend, they will form a cluster and share/distribute state if configured to do so. The server varieties primarily differ within the sort of listener end point accustomed handle incoming connections.

V. CONCLUSION

Infinispan is a data grid platform with the potential to scale to thousands of node. Data in memory is portioned across multiple servers dynamically therefore allows continuous data availability and transactional integrity isn't laid low with server failures.

ACKNOWLEDGMENT

I would like to offer my deepest gratitude to everybody who helped me directly and indirectly for the successful completion of my work. I thank God almighty for all the blessings received during this endeavour. Last, but not the least, I thank all my friends and my family for the support

ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

and encouragement they have given me during the course of my work.

REFERENCES

[1] Rosa, L.; Rodrigues, L.; Lopes, A., \Goal-oriented Self-management of In-memory Distributed Data Grid Platforms", Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on , vol., no., pp.587,591,Nov. 29 2011-Dec. 1 2011
[2] Jing Han, Meina Song, and Junde Song. "A Novel Solution of Distributed Memory NoSQL Database for Cloud Computing", In ICIS 2011, 10th IEEE/ACIS International Conference on Computer and Information Science, 20 I I.
[3] http://www.redhat.com/products/jbossenterprisemiddleware/data-grid
[4] Kai Fan, "Suvey on Nosql", Programmer, 2010(6): pp.76-78
[5] http://www.jboss.org/infnispan/
[6] http://nosql-database.org/
[7] http://infinispan.blogspot.in/2011/02/jsr-107-and-jsr-on-data-grids
[8] https://docs.jboss.org/author/display/ISPN/Cache+Loaders+and+Stores
[9] http://www.jboss.org/rhq
[10] https://www.jboss.org/jbossas

BIOGRAPHIES

Sarith Divakar M received his Bachelor's degree in Information Technology in 2009 from Cochin University of Science and Technology, Cochin. In 2014, he earned Master's degree in Computer Science with specialization in Information Systems from Rajagiri School of Engineering and Technology, Cochin. Currently working as a faculty member at the Department of Computer Science, College of Applied Science, Kozhikode. His research interests include areas like Web services, Data structures, Database systems, NoSQL, Mobile technology. He published various papers in international journals.

Ashina Tholiyil received her Bachelor's degree in Computer Science in 2008 from EMEA College, Malappuram. In 2011, she earned Master's degree in Computer Applications from AWH Engineering College, Kozhikode. Currently working as a faculty member at the Department of Computer Science, College of Applied Science, Kozhikode. Her research interests include areas like Data and System Security, Software Engineering, Software Project Management, Ecommerce systems.