UDC 004.052.3

Yaroslav Hrab,
Artem Volokyta.

IN-MEMORY DATA GRID FAULT TOLERANCE ISSUE BASED ON THE MULTI-MASTER REPLICATION

The paper deals with the issue of fault tolerance of In-Memory Data Grid systems. The fault tolerance mechanism is implemented using the Multi-Master Replication approach. The general structure of the system is presented.

Key words: fault tolerance, Multi-Master Replication, In-Memory Data Grid, transferring data on the cluster.

Fid.: 6. Tabl: 4. Bibl.: 4

Relevance of the research topic. Storing and processing data directly in RAM has been a hotly debated topic in recent times. Many companies that in the past refused to consider the use of in-memory technologies due to their high cost are now rebuilding the architecture of their information systems to take advantage of the fast transactional data processing offered by these solutions. This is due to a sharp drop in the cost of RAM, which makes it possible to store all the entire set of operational data in memory, increasing the processing speed by more than 1000 times compared to data processing on hard drives. In-Memory Data Grid (IMDG) and In-Memory Compute Grid (IMCG) products provide the necessary tools to build such solutions.

One of the key tasks in the development of IMDG system is the organization of fault tolerance of the system. The problem of fault tolerance is due to the fact that the data is stored directly in RAM and any technical problems with the server or power faults can lead to data loss.

Target setting. Lack of a well-described model of the IMGD system, taking into account the need for fault tolerance with minimal data movement within the cluster.

Comparative characteristics of existing solutions. Tables 1 - 3 show the main characteristics of existing IMDG systems in terms of fault tolerance, data security and performance.

Due to Tab. 1, we can see that all compared systems implement replication to save copies of the entire data set. However, it is impossible to say exactly which of the products is the best in terms of fault tolerance, as some of the presented solutions are commercial, therefore it is impossible to get acquainted with the internal structure of these systems.

Due to Tab. 2, we can see that all compared systems implement data access control and transaction concepts. As in the case of fault tolerance comparisons, it is not possible to say exactly in which product data security is better organized.

Table 1

Comparative characteristics of the most popular IMDG in terms of fault tolerance

Name	GridGain	Hazelcast IMDG	NCache	Oracle Coherence	Redis
Replication methods	yes (replicated cache)	yes (Replicated Map)	yes, with selectable consistency level	yes, with selectable consistency level	yes (Master-slave replication; Multi-master replication;)

Table 2

Comparative characteristics of the most popular IMDG in terms of data protection

Name	GridGain	Hazelcast IMDG	NCache	Oracle Coherence	Redis
Access control	Security Hooks	Role-based access control	Active Directory/ LDAP	Authentication to access the cache via certificates or http basic authentication	Simple password-based access control
Transaction concepts	ACID	one or two-phase-commit; repeatable reads; read committed;	optimistic locking and pessimistic locking	configurable	Optimistic locking; atomic execution of commands blocks and scripts;

Table 3

Comparative characteristics of the most popular IMDG in terms of productivity

Name	GridGain	Hazelcast IMDG	NCache	Oracle Coherence	Redis
Concurrency	yes	yes	yes	yes	yes
Durability	yes	yes	yes	yes	yes
In-memory capabilities	yes	yes	yes	yes	yes
XML support	yes	yes	no	no	no
Primary database model	Key-value store; RDBMS;	Key-value store;	Key-value store;	Key-value store;	Key-value store;
Partitioning methods	Sharding	Sharding	Sharding	Sharding	Sharding

Due to Tab. 3, we can see that all compared systems implement the basic

functions that must be present in the IMDG system for it to be possible to effectively interact with it and obtain optimal performance.

Actual scientific researches and issues analysis. At the moment, there is very little scientific research in the public domain that sheds light on this issue. However, there are practical implementations of Multi Master Replication in IMDG systems, such as Redis [1], but this topic is not sufficiently covered in publications.

Uninvestigated parts of general matters defining. This paper is devoted to the study and analysis of the proposed model of the IMDG system, taking into account the need for fault tolerance. The research focuses on the implementation of Multi-Master Replication for IMDG.

Research objective. The objective of this paper is to propose a model of fault-tolerant IMDG system taking into account the minimum amount of data movement within the cluster.

General structure model. Fig. 1 presents a sceme of a three-tier IMDG system with Multi-Master Replication implementation of fault tolerance.

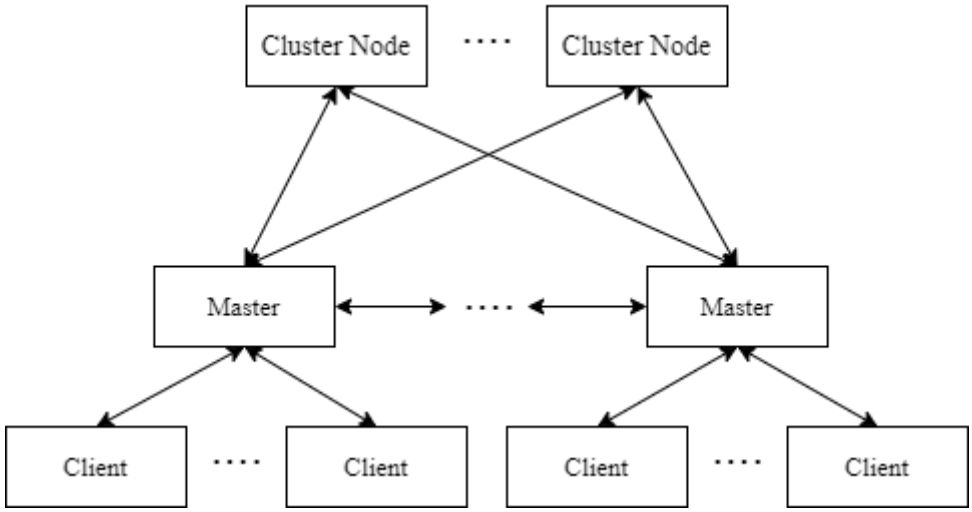

Fig. 1 General model structure

Due to Fig. 1, the Master Tier is an intermediate layer between the client and the server and it is this structure that allows storing the last state of the cache and synchronize it between all components of the Master Tier and the Cluster Tier.

Table 4

Tiers description

Name	Description
Client Tier	The client tier represents the user interface through which the user makes requests to the system.

Ended table 4

Name	Description
Master Tier	This tier is an intermediate layer client and cluster . It receives requests from the client and transmits them to the cluster tier. Its main task is to preserve the latest state of data in the cluster (using non-volatile memory), constant monitoring of the cluster for changes in storage and synchronization between all servers at this tier and all servers at the cluster tier (Delta Tracking).
Cluster Tier	A cluster tier is a distributed in-memory storage. Accepts client requests from master tier and performs them. Also, each cluster-level server stores a copy of the data state in the cluster using non-volatile memory.

Tab. 4 analyzes and describes the purpose of each system level.

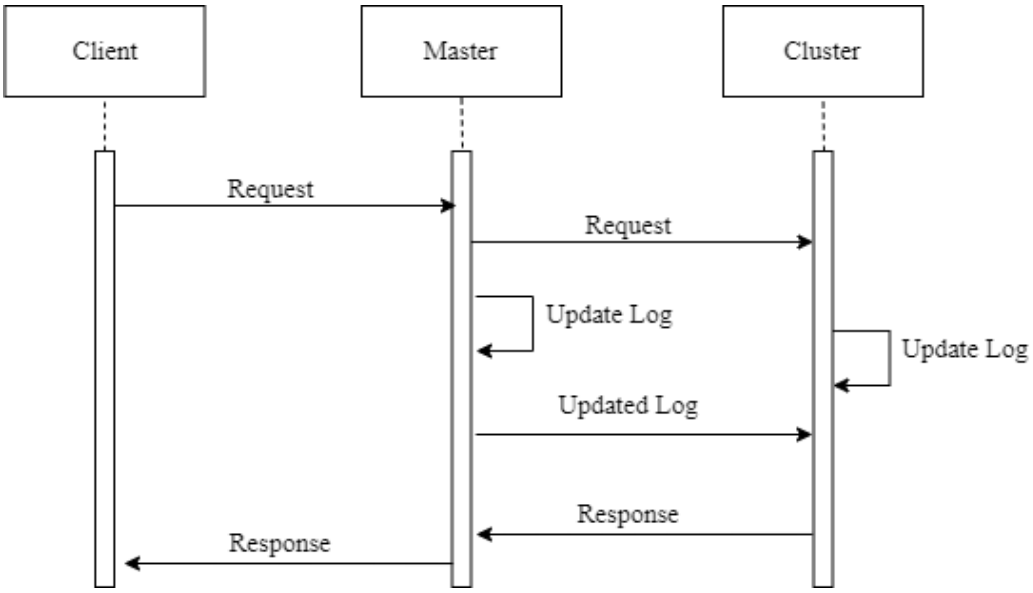

Fig. 2. Algorithm of systems actions when the user adds data to the cluster

Fig. 2 shows how the system creates and sends a copy of the entire data set to its respective components when the cache status is updated.

The following describes the stages of the algorithm shown on Fig. 2:

1. The Master Trier server receives a request from the client and, in accordance with the balancing mechanism, transmits a request to the appropriate cluster server;
2. The Master Tier server updates its file with last state of the cluster;
3. The Master Tier server sends the updated system status file to all other Master Tier servers and all Cluster Tier servers;
4. All servers that received the file update their own files of the latest system state;
5. The Cluster Tier server that executed the client request sends a response to the Master Tier server;
6. The Master Tier server sends a response to the client;

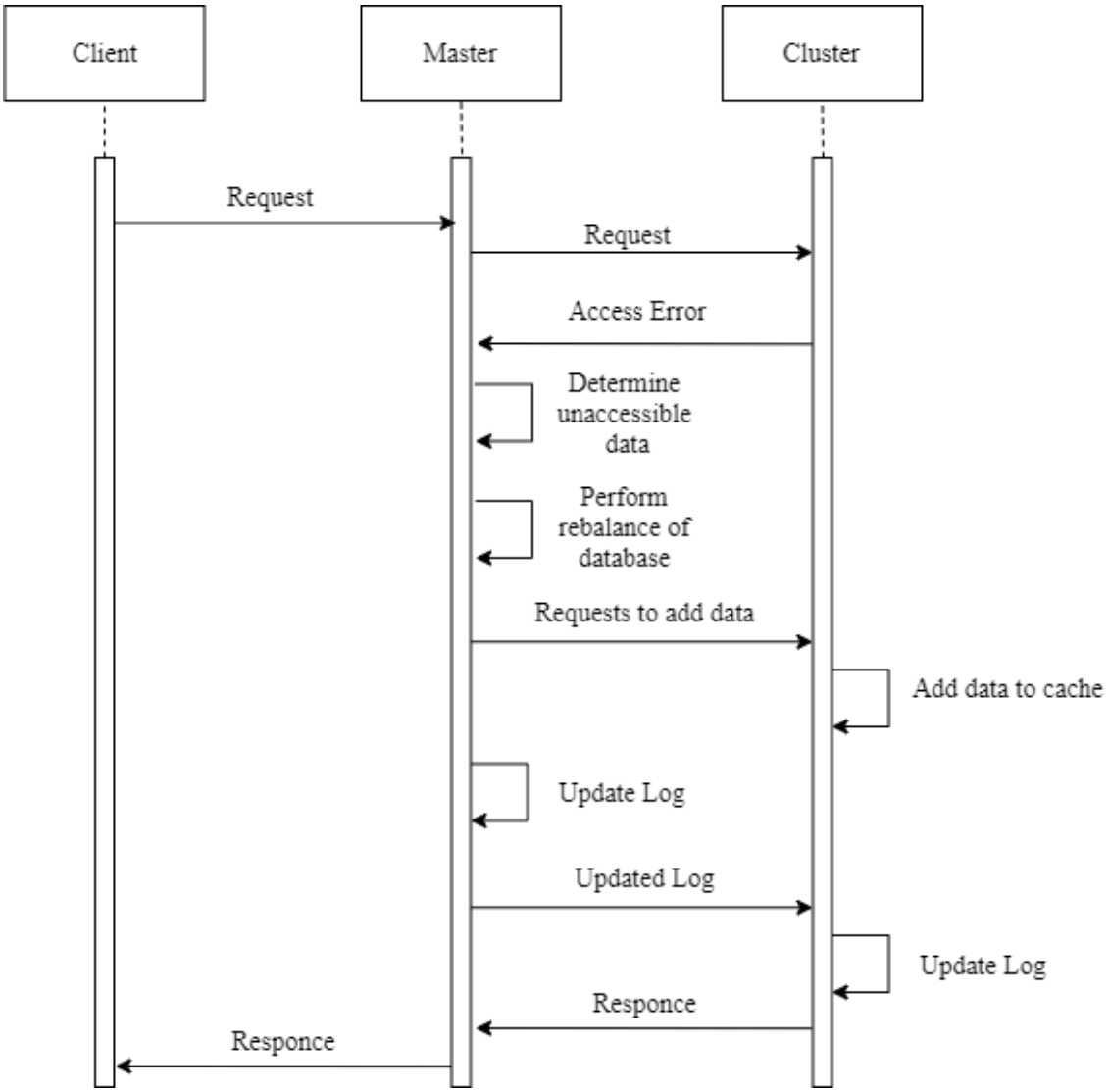

Fig. 3. Algorithm of system actions in case of failure of one of the cluster layer servers

Fig. Figure 3 shows exactly how the system can recover data if one of the cluster servers fails. Using a copy of the entire dataset, the system determines what was on the inaccessible server, balances the system with the new number of available servers, and loads the data into their RAM.

Let's consider the stages of the algorithm shown in Fig. 4:

1. The Mater Tier server detects that one of the cluster servers is not available;
2. The Master Level server determines what data was on the server that is currently unavailable ;
3. According to the balancing mechanism, it is calculated which servers should load the data of the inaccessible server to their RAM;
4. The Master Tier server sends requests to add data to the appropriate servers;
5. Cluster Tier servers add the necessary data to their cache;

6. The Master Tier server updates its file of the last state of the cluster;

7. The Master Tier server sends the updated system status file to all other Master Tier servers and all Cluster Tier servers;

8. All servers that received the file update their own files with the latest system status;

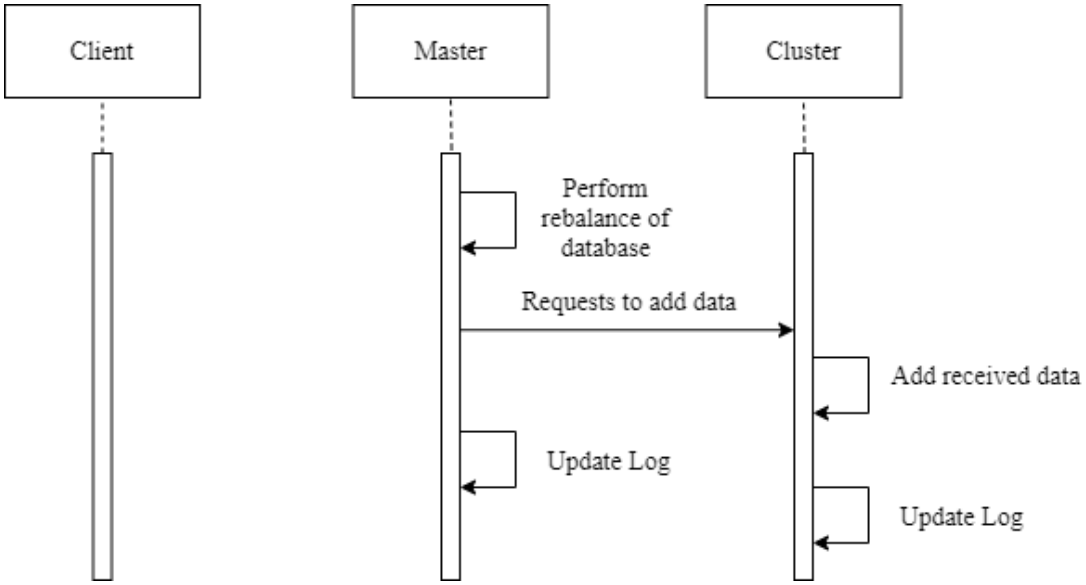

Fig. 4. Algorithm for restoring the state of the system after rebooting the cluster

Fig. Figure 4 shows exactly how the system can restore its previous state after a scheduled or unplanned shutdown of all cluster servers. This approach allows the maintenance of the entire cluster without the risk of losing cached data.

Let’s consider the stages of the algorithm shown in Fig. 4:

1. The Master Tier server determines what data was on each cluster server;
2. The Master Tier server sends requests to add data to the appropriate servers;
3. Cluster Tier servers add the necessary data to their RAM;
4. The Master Tier server updates its file of the last state of the cluster;
5. The Master Tier server sends the updated system status file to all other Master Tier servers and all Cluster Tier servers;
6. All servers that received the file update their own files with the latest system status;

The main advantage of the presented model is the absence of data movements within the cluster. If data were moved directly between cluster servers, this would necessitate rebalancing the cluster after each such move. And this in turn leads to a significant reduction in the speed of the entire system.

Example of system operation. Fig. 5 - 6 shows a fragment of the software implementation of the request handler and an example of interaction with the system.

```
ссылка: 1
public string PerformRequest(Command command) =>
    command.Kind switch
    {
        CommandKind.Get => PerformGet((GetCommand) command),
        CommandKind.Set => PerformSet((SetCommand) command),
        CommandKind.GetAll => PerformGetAll((GetAllCommand) command),
        CommandKind.Keys => PerformKeys((GetKeysCommand) command),
        CommandKind.Update => PerformUpdate((UpdateCommand)command),
        CommandKind.Remove => PerformRemove((RemoveCommand) command),
        CommandKind.Find => PerformFind((FindCommand) command),
        CommandKind.Clear => PerformClear((ClearCommand) command),
        CommandKind.HSet => PerformHSet((HSetCommand) command),
        CommandKind.HGetAll =>PerformHGetAll((HGetAllCommand) command),
        CommandKind.HGet => PerformHGet((HGetCommand)command),
        CommandKind.HVal => PerformHVal((HValCommand)command),
        CommandKind.HRemove => PerformHRemove((HRemoveCommand)command),
        CommandKind.HKeys => PerformHKeys((HKeysCommand)command),
        CommandKind.LAdd => PerformLAdd((LAddCommand)command),
        CommandKind.LGetAll => PerformLGetAll((LGetAllCommand)command),
        CommandKind.LRemove => PerformLRemove((LRemoveCommand)command),
        CommandKind.LCount => PerformLCount((LCountCommand)command),
        _ => "Invalid command"
    };
```

Fig. 5. A fragment of the software implementation of the command handler in .NET 5 environment

```
Connected to the server!
Enter the request:
hset persons Smith John
Sended: hset persons Smith John
Answer:
Smith John pair has been added to hash map associated with key persons

Enter the request:
hset persons Web Ian
Sended: hset persons Web Ian
Answer:
Web Ian pair has been added to hash map associated with key persons

Enter the request:
hgetall persons
Sended: hgetall persons
Answer:

(Web Ian)
(Smith John)

Enter the request:
hval persons
Sended: hval persons
Answer:

(Ian)
(John)

Enter the request:
```

Fig. 6. Example of client interaction with the system

As seen in Fig. 5, the command handler receives the user's request, determines the type of request, checks the validity of the argument and executes it. If the received command does not exist, the handler returns the corresponding message.

As seen in the Fig. 6, the client part of the system is presented as a console

application. The user enters the query, that query is then checked by the command handler and executed. The result is displayed on the user's console.

Conclusions. The paper demonstrates a general model of an IMDG system with a fault tolerance mechanism based on Multi-Master Replication. The proposed model eliminates the need to move data within the cluster, which in turn does not reduce performance.

There are several areas for further work. One of them is the development of interaction protocols between all levels of the system. In addition, further research in the field of developing efficient algorithms for multi-server requests is possible and should be considered. These changes will significantly increase the efficiency of the system.

REFERENCES

1. Sanfilippo S. Redis, 2015. URL: https://github.com/antirez/redis
2. System Properties Comparison GridGain vs. Hazelcast vs. NCache vs.Redis. URL: https://db-engines.com/en/system/GridGain%3BHazelcast%3BNCache%3BRedis
3. Ivanov N. GridGain, In-Memory Data Grid: Explained..., 2012. URL:
4. https://www.gridgain.com/resources/blog/in-memory-data-grid-explained
5. Репликация данных. URL: https://ruhighload.com/%D0%A0%D0%B5%D0%BF%D0%BB%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D1%8F+%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85

AUTHORS

Yaroslav Hrab – 4th year student, Department of Computer Engineering, National Technical University of Ukraine “Igor Sikorsky Kyiv Polytechnic Institute”.

Email: jkjkjmenq@gmail.com

Artem Volokyta (supervisor) – associate professor, Department of Computer Engineering, National Technical University of Ukraine “Igor Sikorsky Kyiv Polytechnic Institute”.

E-mail: artem.volokita@kpi.ua