Tensor: A Transaction-Oriented Low-Latency and Reliable Data Distribution Scheme for Multi-IDCs Based on Redis

Zhongyi Zhang
Institute of Information Engineering
Chinese Academy of Sciences
& School of Cyber Security
University of Chinese Academy of Sciences
Beijing, China
zhangzhongyi@iie.ac.cn

Chao Zheng
Institute of Information Engineering
Chinese Academy of Sciences
Beijing, China
zhengchao@iie.ac.cn

Wei Yang
Institute of Information Engineering
Chinese Academy of Sciences
Beijing, China
yangwei@iie.ac.cn

Yang Liu
Institute of Information Engineering
Chinese Academy of Sciences
Beijing, China
liuyang@iie.ac.cn

Rong Yang*
Institute of Information Engineering
Chinese Academy of Sciences
Beijing, China
yangrong@iie.ac.cn

Qingyun Liu
Institute of Information Engineering
Chinese Academy of Sciences
Beijing, China
liuqingyun@iie.ac.cn

Abstract—In order to ensure the data security, high availability of services and good access performance, more and more large-scale distributed systems are deployed across IDCs. When different parts of distributed systems work cooperatively, critical data such as configuration and control information will be frequently exchanged across IDCs.

Faced with the latency and reliability challenges introduced by cross-IDC data distribution. We propose a Redis-based low-latency data publish/subscribe framework: Tensor. In order to deal with the problems of data loss and duplication caused by network anomalies or cluster node failures, we design a transaction-oriented information transfer mechanism in Tensor to guarantee the eventual consistency in cross-IDC data distribution. To improve the data synchronization performance, we optimized Redis's replication mechanism to make it better suit the unstable network links between cross-area IDCs. What's more, we design an intelligent log analysis based system bottleneck prediction method and a service discovery oriented system failover strategy to ensure the high availability of data distribution service. An extensive set of tests on Tensor in the production environment prove the low-latency and high-reliability of its data distribution service.

Index Terms—cross-IDC data distribution, transaction-oriented, Replication Backlog, intelligent log analysis

I. Introduction

With the rapid development of the Internet scale and the diversification of Internet services, single server can no longer support the need of mass data storage and highly concurrent user access, so, distributed systems has been applied more and more widely. In order to ensure high availability of services as well as improve systems' performance, more and more large-scale distributed systems are deployed across IDCs. When different parts of distributed systems work cooperatively, critical data such as configure and control information should be frequently exchanged across data centers.

Corresponding author: yangrong@iie.ac.cn

The critical data transmitted between different parts of distributed systems is very sensitive to latency. However, the deploying scenarios of cross-IDC distributed system pose a significant challenge to low-latency data distribution between different inner components. Firstly, for the consideration of data security, IDCs are usually located in different areas, which means high network transmission latency. Secondly, information transmission between different data centers requires multi-hop routing, routing selection and switching will lead to higher data processing delay. Thirdly, in cross-area network environment, the contention for link bandwidth is serious, which is a barrier to fast data transmission.

For key information with high value and sensibility, cross-IDC data distribution service must be highly reliable while ensuring low latency. However, the reliability of data distribution service also faces a number of challenges: Above all, the underlying physical nodes of the data distribution service are at risk of failure. According to the report of Facebook, the Hadoop cluster consisting of 3000 nodes has an average of 22 node failures per day, with the highest number of node failures exceeding 100 per day [1]. Furthermore, fire, earthquake, thunder and other irresistible factors will seriously threaten the normal operation of IDC rooms, and further damage the reliability of data distribution service. Last but not least, cross-area network links are unstable which will cause network congestion and packet loss thus threaten the normal transmission of data.

Therefore, in the scenario of cross-IDC deployment of distributed systems, it's of great significance to study how to effectively reduce critical data's distribution latency and improve the reliability of data distribution service. In this paper, we propose a Redis-based low-latency data publish/subscribe framework: Tensor. We use Redis [2] which is an open source, in-memory NoSQL database to store and distribute critical data between different cross-IDC components in a large distributed system. The K/V data storage method of Redis can well meet the processing requirements of configure and control information. What's more, Redis provides flexible master-slave synchronization method with full-scale and incremental replication function which is friendly for data distribution in unstable network environment. In order to deal with the problem of data loss and duplication caused by network anomalies or cluster node failures, we design a data-consistent information transfer mechanism in Tensor to guarantee the eventual consistency in cross-IDC data distribution. And then, to improve the data synchronization performance, we optimized Redis's replication mechanism to make it better suit the unstable network links between cross-area IDCs. Finally, we design an intelligent log analysis based system bottleneck prediction method and a service discovery oriented system failover strategy to ensure the high availability of data distribution service.

The rest of the paper is organized as follows: Section II discusses the related works. Section III presents the system architecture of Tensor. Section IV elaborates the specific method designs. Section V introduces the extensive set of tests in the production environment. And finally the paper is concluded in Section VI.

II. Related Work

This section mainly introduces the domestic and foreign situations towards the study of the data distribution middleware both in academia and industry.

In academia, HBaseMQ [3] is the first advanced message queue based on the HBase Cloud, it supports "at least once" or "at most once" message delivery semantics, and has no limitation on the message size. However, HBaseMQ is highly coupled with Hadoop/HDFS ecosystem and is not suitable for the data distribution service in general scenarios. HDMQ [4] uses hierarchical distributed message queues and supports the data transmission in cross-domain scenarios, it guarantees the time-sequence and "exact once" semantics of data delivery, the size of the message in it is limited to no more than 512KB. FaBRiQ [5] is based on DHT(distributed hash table), in which the P2P data transmission strategy is used to ensure the flexibility and scalability of the Broker cluster, it is suitable for the data distribution scenarios where the disordering of messages is allowed. RDDS [6] relies on the publish/subscribe model, it can maintain the robustness, efficiency and consistency of the data distribution service under unpredictable workloads, its main application area is the scenarios of data transmission between entities with small spatial span. CoreDX DDS [7] is compatible with the OMG DDS standard [17], it can meet the requirements of "getting the right data at the right time and right place" and mainly used in embedded systems. CoreDX DDS chiefly focuses on improving the overall performance of the data distribution service, its reliability is relatively low.

In industry, Apache Kafka [8] is an open source message distribution middleware with high throughput. It guarantees the order of messages within a partition, but the message sequence between partitions cannot be ensured. Kafka supports "at least once" or "at most once" message delivery semantics, and can adapt to the growing amount of data by increasing the number of Brokers horizontally, in cross-area data distribution scenarios, its latency will be relatively high. RocketMQ [9] emerges by the driving of Alibaba's specific business needs. But in pursuit of the high throughput and low latency of message transmission, RocketMQ abandons the high reliability of its data distribution service. Amazon SQS [10] [11] [12] is a message-oriented middleware widely used in business at present which is simple, safe and performs well in the aspect of scalability and service availability, it ensures the "at least once" message delivery semantic, and the size of the message is limited to no more than 512KB. Tencent CMQ [13] supplies distributed message queuing services with high reliability and great performance. The ability of elastic expansion and massive data stacking is provided as well. However, it uses the cold standby method for fault tolerance which will take relatively longer time for service discovery. Dragonfly [14] is a general data distribution system based on intelligent P2P technology which can solve challenges in the large-scale file distribution scenario, such as problems of the high time consumption, low success rate, bandwidth waste and so on, it is mainly applied to transmit large files such as containers and mirrors.

In summary, to the best of our knowledge, no matter in academia or industry, existing middlewares cannot simultaneously meet the high availability and low latency requirements of the large-scale cross-IDC data distribution service in general scenarios.

III. System Architecture Overview

A. Basic Architecture

Faced with the cross-area data distribution scenario, we implemented a Redis-based low-latency data publish/subscribe framework with hierarchical architecture called Tensor.

The basic architecture of Tensor is shown as "Fig. 1" and "Fig. 2", it composes of three different parts: Data Publisher, Data Subscriber and Broker. The Data Publisher/Subscriber is responsible for the interaction with the Data Producer/Consumer, and the Broker refers to the inner physical nodes of our data distribution cluster. We divide the Broker nodes into two tiers, and all Broker nodes at the same tier form a BrokerSet [15] [16]. Multiple Redis instances are started on each Broker, one of which is the master and the others are slaves. The terms in Tensor are explained in "Table I".

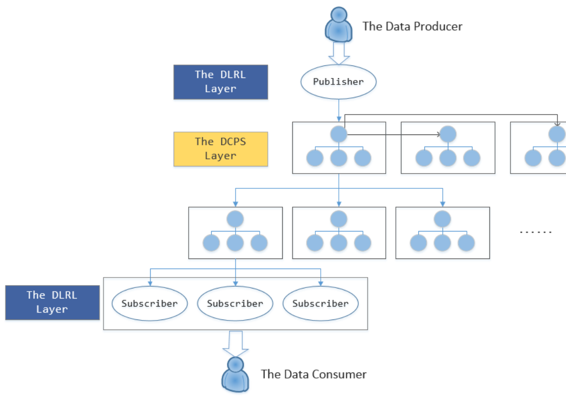


Fig. 1. The basic architecture of Tensor: the DCPS layer and the DLRL layer.

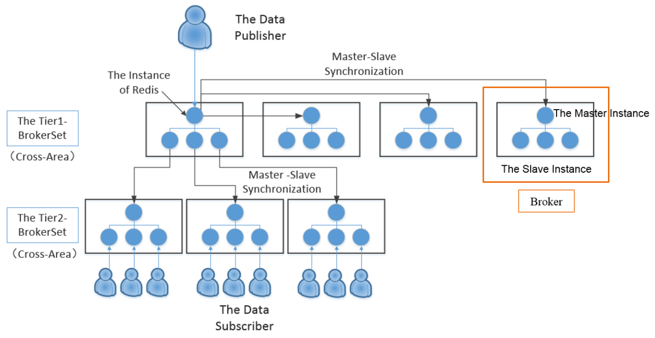


Fig. 2. The basic architecture of Tensor: the Tier-1 BrokerSet and the Tier-2 BrokerSet.

TABLE I
The interpretation of terms in Tensor

Terms	Interpretations	Businesses
Publisher	The data publisher	Produce and send data
Subscriber	The data subscriber	Receive and consume data
Broker	The data storage and distribution node	Receive, store and distribute data
BrokerSet	The cluster of brokers	A cluster of brokers, usually deployed in different IDCs

We follow the OMG DDS [17] specification, which divides the data distribution service into two distinct layers, the low-level DCPS(data-centric publish/subscribe) layer and the high-level DLRL(data local reconstruction) layer. Among them, DCPS is responsible for the efficient and accurate transmission of information from data publishers to data subscribers, DLRL abstracts the functions provided by DCPS and establishes the mapping relationship with the underlying services. In Tensor, the Broker nodes constitute the low-level DCPS layer, while the Data Publishers/Subscribers constitute the high-level DLRL layer.

B. Module Design

In the specific module design aspect, Tensor consists of three different tiers: the Application Tier, the Management Tier and the Service Tier, as shown in "Fig. 3".

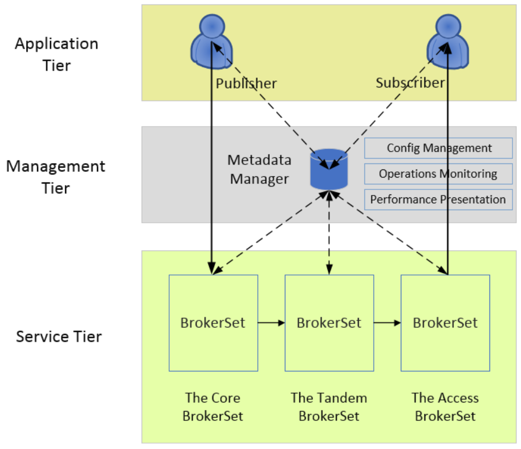


Fig. 3. The module design of Tensor.

In the Application Tier, we defined a set of data communication interfaces between subsystems of data distribution business. What's more, we provide lightweight clients that include publishing and subscribing interfaces for multiple types of data.

In the Management Tier, Data Publishers/Subscribers and BrokerSets establish long connections with Managers, and send heartbeats regularly, so that Managers can monitor the status and health of Tensor. In addition, this layer also supports abnormity alarm which can ensure the stable operation of the whole system indirectly. Last but not least, real-time status of the system's performance is displayed, including average or variance of data distribution time, system's link topology information, the running states of each node, and so on.

The Service Tier is the core of our data distribution service, we design three different kinds of BrokerSets to meet the requirements of cross-IDC data distribution service, which are defined as below:

- The Core BrokerSet: The Core BrokerSet plays the part of bridges between Data Publishers and the Tandem BrokerSet.
- The Tandem BrokerSet: In order to distinguish the Core BrokerSet which connects to Data Publishers and the Access BrokerSet which connects to Data Subscribers, we named the BrokerSet in the middle

layer as Tandem BrokerSet which can be arbitrarily extended. If the number of Data Subscribers or the network scale is small, Tandem BrokerSet may not be deployed.

- The Access BrokerSet: The Access BrokerSet plays the role of bridges between the Tandem BrokerSet and Data Subscribers.

Due to the hierarchical design of the architecture, Tensor supports both scale-out and scale-up. When a single data center cannot meet the demands for enterprise scale, we can horizontally add BrokerSets within or outside the same city to achieve horizontal expansion. If the subscribers' number or network scale increases, and data needs to be transmitted across areas or even across network operators, we can extend the Tandem BrokerSets vertically to meet the above requirements.

IV. Specific Method Design

A. A transaction-oriented data publish/subscribe approach

In order to avoid data inconsistency between subscribers and publishers due to node, network or IDC anomalies, we design an asynchronous transaction-based data publish/subscribe approach. Every single step of information transmission is strictly protected by the transaction mechanism on the foundation of Redis.

The basic architecture of our approach is shown in "Fig. 4", we support multiple businesses, each of them corresponds to a database in Redis respectively, and the isolation between different businesses will be severely ensured. "Table II" shows the specific data structures we designed to guarantee the eventual data consistency between Data Publishers and Subscribers, and we store the data structures in the Redis component.

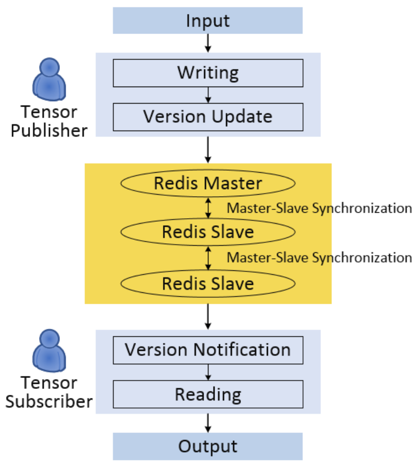


Fig. 4. The basic architecture of an asynchronous transaction-based data publish/subscribe approach.

On the Data Publisher side, each manipulation activity to our database is recorded by a global unique version id, namely Tensor_VERSION in "Table II". The manipulation commands are divided into two different types: addition commands and deletion commands, the format of the two kinds of commands are "ADD, ID" and "DEL, ID" respectively, in which "ADD" and "DEL" indicate data operations, and "ID" uniquely targets one specific piece of data. Several commands form one manipulation activity and every single manipulation activity is protected by the transaction mechanism of Redis, which ensures the integrity and time sequence of the data in the same batch. The manipulation commands are stored as the member part in a sorted set in Redis which named Tensor_UPDATE_STATUS, and the version id of each manipulation activity mentioned above is stored as the score part of Tensor_UPDATE_STATUS. Every time, after we publish a set of manipulation commands to Tensor as the same batch, the Tensor_VERSION will be increased by one and the manipulation activity will be uniquely recorded by this version id.

TABLE II
The specific data structures of the asynchronous transaction-based data publish/subscribe approach

Redis Key	Explanation	Structure
Tensor_VERSION	Version ID	Integer
Tensor_UPDATE_STATUS	The update status of data	Sorted set
EFFECTIVE_RULE:ID	Effective data	String
OBSOLETE_RULE:ID	Obsolete data	String
Tensor_EXPIRE_TIMER	Timeout information	Sorted set

On the Data Subscriber side, each Subscriber maintains a version id locally which is called Sub_VERSION, and the subscription behavior is driven by a trigger we name it Data_Traverser: a separate monitor which compares Sub_VERSION and Tensor_VERSION periodically, once Data_Traverser finds the local Sub_VERSION lags behind the global Tensor_VERSION, the Subscriber will be triggered to subscribe the newest manipulation commands from Tensor, more specifically, we use ZRANGEBYSCORE command to get the data whose version ID falls between Sub_VERSION and Tensor_VERSION from the sorted set named Tensor_UPDATE_STATUS in Redis. Every time when a Subscriber gets a full version of data from Tensor, its local Sub_VERSION will be added by one. The publication/subscription of manipulation commands as well as the increase of Sub_VERSION and Tensor_VERSION are all severely protected by transaction, and the duplication or loss of valuable data will be strictly prevented.

In order to ensure Tensor's low consumption of memory space, each manipulation command is set a certain life cycle, the precise expiration timestamp of each command can be calculated by using this certain life cycle add the exact timestamp when data is published. We use another sorted set in Redis named Tensor_EXPIRE_TIMER to store the above information, of which the member part is the manipulation commands and the score part is their corresponding expiration timestamp. The expired commands until this very moment will be monitored by a separate trigger named Data_Scavenger, which will periodically use ZRANGEBYSCORE command to get the members in Tensor_EXPIRE_TIMER whose expiration timestamp is between negative infinity and the current moment, and then the corresponding keys will be set ineffective using EXPIRE command. Redis will clear these ineffective keys after a certain period of time.

B. An efficiency-guaranteed data distribution mechanism

In the production environment, Tensor performs dozens to hundreds of data distribution tasks every day, several of them are the mission to transmit large files of data types, the size of a single file is around 1.5GB, the remaining tasks are to transmit configuration or control type messages, the total data amount of s single task ranges from several hundred KBs to a few MBs.

In Tensor, the efficiency of data distribution is largely determined by the performance of Redis's master-slave synchronization. The reason for Redis's execution of data synchronization is the inconsistent status between master and slave servers. Most of the time, the "Command Propagate" method [2] is used by Redis to synchronize data, in which case, the commands propagated from the master sever will not only be received and executed by all slave servers, they will also be inserted into a circular queue named "Replication Backlog", as shown in "Fig. 5". The space of Replication Backlog is limited, after all of which been occupied, the old data will be covered from the Head pointer, so only the newest commands will be kept in Replication Backlog.

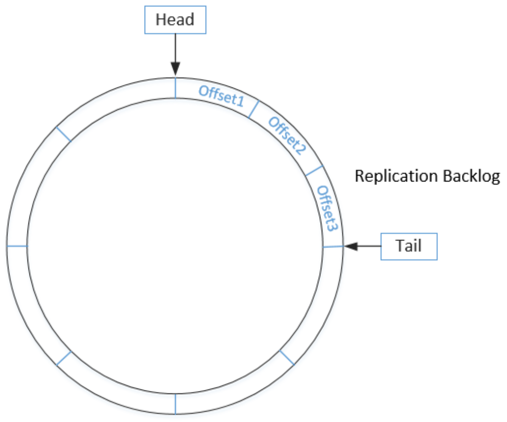


Fig. 5. The basic structure of the Replication Backlog in Redis.

In the unstable network environment between cross-area IDCs, the occurrence of the master-slave connection's abnormality is almost inevitable. Every time when broken connections between master and slave servers are restored, the Offset values maintained by slaves will be sent to the master server, we denoted them as Offset_Recv. If the write commands executed by the master server whose Offset value is larger than Offset_Recv are all still remained in Replication Backlog, they will be sent to slaves directly, otherwise, full resynchronization is needed.

In order to avoid the full data synchronization of Redis to the greatest extent in cross-IDC network environment, we design a dynamic Replication Backlog adjustment method based on exponential backoff strategies.

We denote the size of Redis's Replication Backlog as R_B_Size and set its initial value to 10MB(Exceed the maximum amount of data size for a single transmission task of small messages). The real-time data write rate of the Master server was monitored by a separate trigger we name it Buffer_Regulator, meanwhile, the average duration time of each disconnection event between master and slave servers in the last 24 hours will be calculated and recorded, we call it Aver_Disconnect_Time.

The trigger Buffer_Regulator will compute the product of the real-time data write rate and Aver_Disconnect_Time periodically in every 30 seconds and we denote the result as Prediction_Space_Size, if the value of Prediction_Space_Size is less than the current size of Replication Backlog, no operation is required, otherwise, the size of Replication Backlog will be instantaneously raised to Prediction_Space_Size.

Most of the time(when Tensor performs the task of distributing small messages such as configure or control information), the data writing rate of the master server is very small, to avoid the waste of the servers' memory resources, we will reduce the space occupation of Replication Backlog to a lower level(10MB). The formula of the exponential model is shown as follows, in which t represents the time interval since the nearest event for the space raise of Replication Backlog:

$$R\_B\_Size = \begin{cases} 10, & if\ R\_B\_Size * 2^{-t} < 10; \\ R\_B\_Size * 2^{-t}, & else. \end{cases} \tag{1}$$

"Fig. 6" shows the process of the dynamic adjustment of Replication Backlog mentioned above.

C. A multi-strategy-integrated high reliability assurance method

Faced with large-scale data distribution scenarios across IDCs and regions, the service availability of Tensor is confronted with severe challenges. To ensure the high reliability of the whole system, when exceptions of system components happen, the failover operations must be conducted in time. However, in large-scale distributed systems, there are often a series of obvious characteristics before system components fail, such as the abnormality of heartbeats, the high latency of network links, the long-term overload of CPU, memory and disk, etc. With

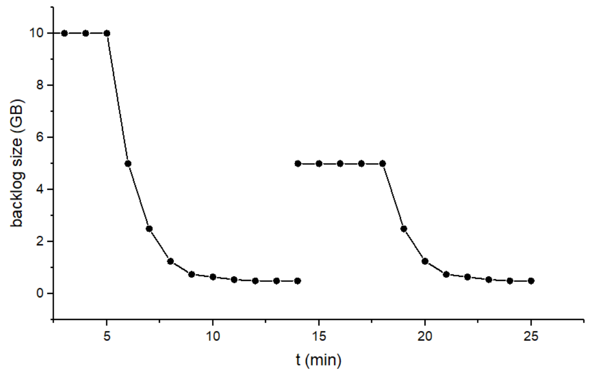


Fig. 6. A dynamic Replication Backlog adjustment method based on exponential backoff strategies.

these features, the exceptions of system components can be predicted. When conducting failovers, the choice of quasi-failure components to take over the job should be consciously avoided. To the best of our knowledge, the field mentioned above has not been fully explored.

In this paper, we use the method of intelligent log analysis to predict the health status of physical nodes, use the approach based on service discovery to conduct failovers, and the predicted health status of each node will be used as strategic supports for failovers to ensure the high reliability of Tensor system.

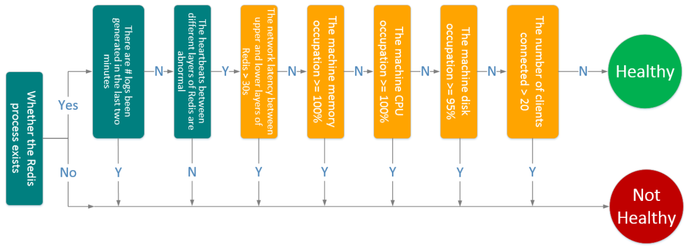
We collect eight different kinds of information in the logs of each physical node in Tensor system, The Specific details and threat level of them to the reliability of Tensor system is shown in ”Table III”.

TABLE III
Selected log information and its threat to the reliability of Tensor system

The Log information	The threat level
The Redis process is crashed	Extremely high
There are # logs been generated in the last two minutes	High
The heartbeats between different layers of Redis are abnormal	High
The network latency between upper and lower layers of Redis > 30s	High
The machine memory occupation >= 100%	Relatively high
The machine CPU occupation >= 100%	Relatively high
The machine disk occupation >= 95%	Relatively high
The number of clients connected > 20	Relatively high

We use the decision tree model shown in “Fig. 7” to predict the health status of each node. In this decision tree, the factors which are closer to the root are more harmful to the reliability of Tensor system.

If one of the above eight kinds of information appears in the log of a physical node in the last 30 minutes, we



Whether the Redis process exists
Yes
No
There are # logs been generated in the last two minutes
The heartbeats between different layers of Redis are abnormal
The network latency between upper and lower layers of Redis > 30s
The machine memory occupation >= 100%
The machine CPU occupation >= 100%
The machine disk occupation >= 95%
The number of clients connected > 20
N
Y
Healthy
Not Healthy

Fig. 7. The node health state prediction method based on intelligent log analysis.

will judge this node to be sub-healthy and reduce its alternative priority in the failover operation.

The rest of this chapter will be arranged to introduce our failover strategy of Tensor. The strategy is divided into two different Tiers, in which Tier1 and Tier2 are oriented to data publishing and subscribing services, respectively. The overall architecture of our high availability guarantee scheme is shown in “Fig. 8”, and Consul is used as the basic component of service discovery.

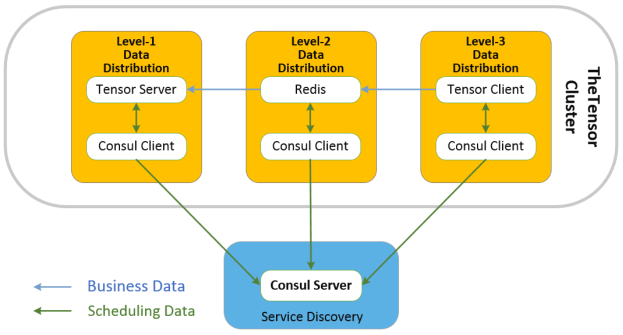


Fig. 8. The cluster’s high availability guarantee scheme based on service discovery.

The basic principle of Tier1-level high availability assurance scheme is shown in “Fig. 9”, our failover system will monitor all nodes in the Tier1-BrokerSet, once the duration time of the master server’s offline state exceeds our threshold(D-J-Threshold), the election procedure will be activated, it consists of three steps:

- Save all slave servers into a list;
- Remove sub-healthy nodes determined by the decision tree mentioned above from the list;
- Select the slave server with the largest replication offset(which is the server that holds the latest data) from the remaining servers in the list.

Then our failover system will upgrade the selected server to be the new master and set the remaining servers as its slaves.

As shown in “Fig. 9”, the Consul Cluster will conduct health checks on all Redis nodes periodically in every 2 seconds, and all the changes of the master-slave relationship will be perceived and reserved in time. When the Data Publisher publishes data, it will firstly send HTTP

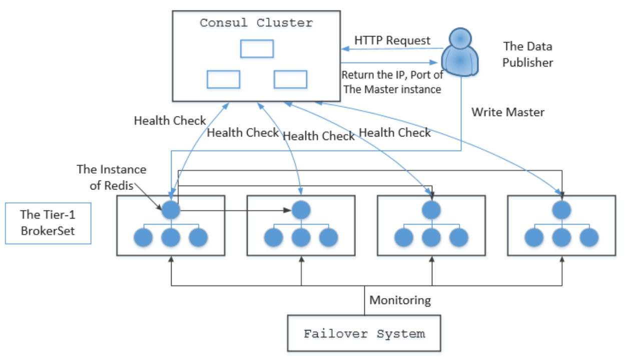


Fig. 9. The Tier1-level high availability guarantee scheme.

requests to the Consul Cluster, regardless of whether or not node failures occur on the Tier1-BrokerSet, the IP and PORT of the latest service-available master will be returned and the high availability of the data publishing service will be strictly ensured.

The basic principle of Tier2-level high availability assurance scheme which is responsible for the data subscribing service is shown in "Fig. 10", due to space constraints, we are not going to detailly describe it here.

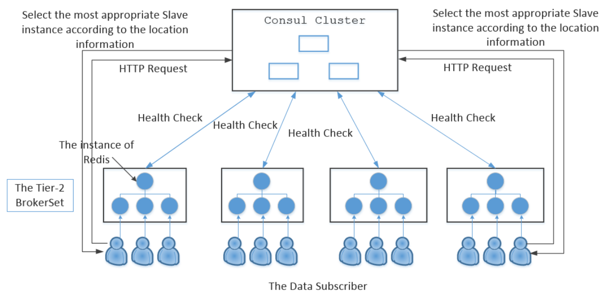

Select the most appropriate Slave instance according to the location information
Consul Cluster
Select the most appropriate Slave instance according to the location information
HTTP Request
HTTP Request
Health Check
Health Check
Health Check
Health Check
The instance of Redis
The Tier-2 BrokerSet
The Data Subscriber

Fig. 10. The Tier2-level high availability guarantee scheme.

V. Evaluation

We conduct an extensive set of tests on Tensor in the production environment to evaluate its data distribution service from different perspectives, including the performance, throughput, scalability, load balancing effects and reliability, etc.

A. The experimental environment

We deploy the Tensor system in hundreds of ISP IDCs located in three different areas: A, B and C. According to the hierarchy, these IDCs are divided into three different levels: the national headquarter, divisions of the A, B, C area and their subordinate points(domains), in which deployed the master server, Tier1-Brokerset, Tier2-Brokerset of Tensor, respectively, as shown in "Fig. 11" and "Fig. 12". After our data is written to the master, Tensor will distribute them from the national headquarter to all subordinate levels in turn. At the bottom of our topology, the number of data subscribers is around 1.0w.

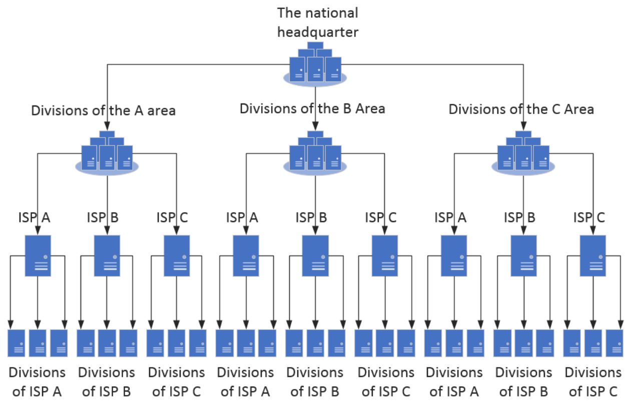


Fig. 11. The topology of the IDCs in our production environment.

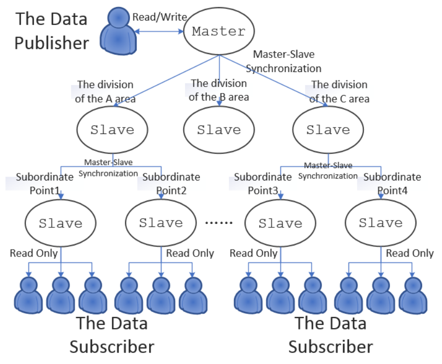


Fig. 12. The topology of Tensor in the production environment.

At the same time, we deployed a large-scale Consul cluster online. To ensure the high availability of service discovery functions, we provide four IDCs for the server side of Consul, namely the IDCs of the national headquarter and the divisions of the A, B, C area, as shown in Figure13.

The hardware and software configurations of the servers is shown in "Table IV", and the version of Redis is 4.0.11.

TABLE IV
The basic software and hardware configuration of test servers

Basic software & hardware	The configuration information
CPU	Intel(R) Xeon(R) CPU E5-4620 0 @ 2.2GHz
Memory	DDR3 SDRAM 128G
NIC	Broadcom Corporation NetXtreme BCM5720 Gigabit
Operating System	Red Hat Enterprise Linux Server release 6.4 Santiago
Kernel	Linux 3.10.0-327.el7.x86_64

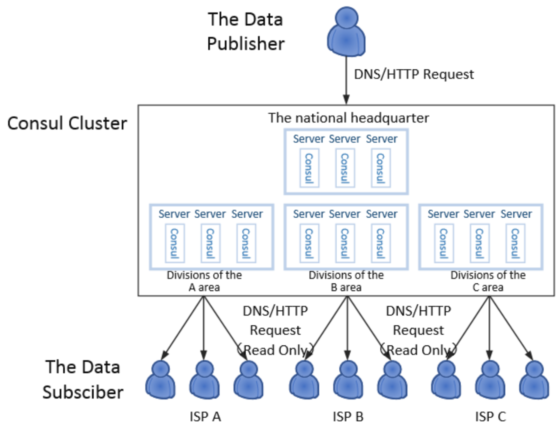


Fig. 13. The topology of the Consul clusters in the production environment.

B. System Performance & Throughput

First of all, we measure the network latency between the headquarter and divisions of A, B and C area, which is 25ms, 30ms and 32ms, respectively. We fix the message size to 1KB, with the number of messages increases from 1 to 1000w, the time consumed by the distribution of our data from national headquarters to all subscribers is shown in Figure 14. Experiment results show that, when the number of messages is less than 5w, the data distribution time is about 12s, and when the message number exceeds 5w, the data distribution time starts to increase, which is shown in “Fig. 14”.

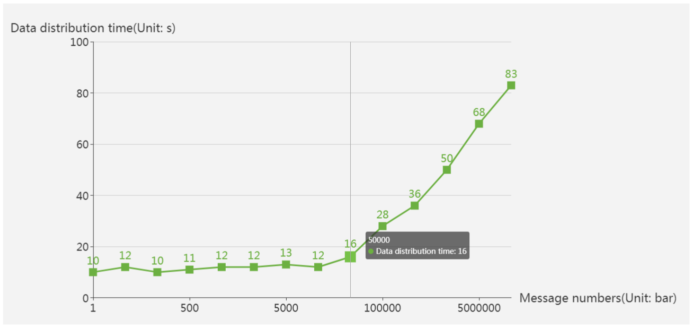


Fig. 14. The test on system performance and Throughput.

C. System Scalability

We decrease the size of Tensor’s subscribers from 1.0W to 20 in turn, and the number of messages is fixed to 1000 and 500W, respectively. The data distribution time from national headquarters to all subscribers is shown in “Fig. 15” and “Fig. 16”. As shown in the picture, when the number of messages is fixed at 1000, the data distribution time of the whole network consumes about 12 seconds and is relatively stable. At the mean time, while the number of messages is fixed at 500W and subscribers’ size is less than 600, the data distribution time is around 50 seconds, furthermore, while subscribers’ size exceeds 600, the data distribution time starts to increase.

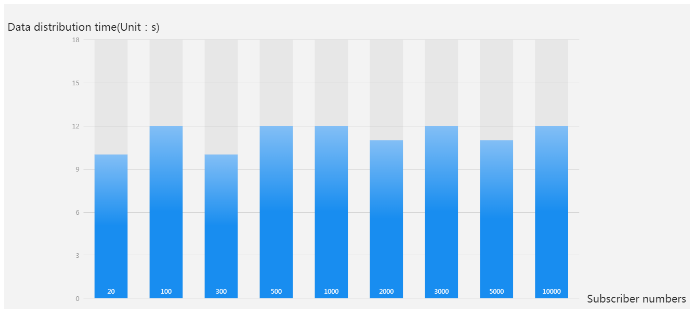


Fig. 15. The test on system scalability(Message number is fixed to 1000).

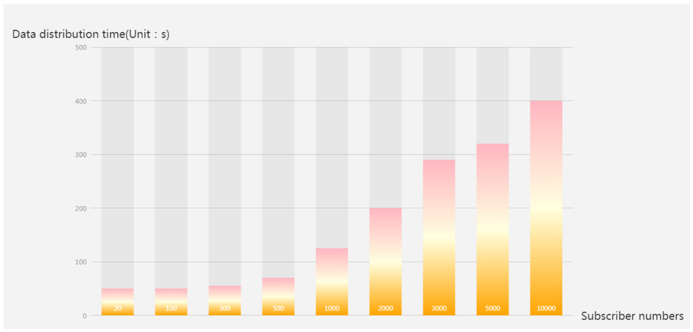


Fig. 16. The test on system scalability(Message number is fixed to 500w).

D. System Load Balancing Effects

To evaluate the load balancing effects of Tensor, the standard deviation of the message numbers on different brokers will be monitored, calculated and presented instantly. Every time when a new batch of data is published, there will be a spike which represent the standard deviation value on the dashboard, and when there are no more new data being published, the spikes will fade away in 20 seconds (For the consideration of the sensitivity of this statistical data, the detailed figures are now listed here).

E. System Reliability

The seasonal reliability test of the whole Tensor system has been conducted twice and it will become normalized. In the first test, all the IDCs in the divisions of the A area are cut off the electricity supply. Within 30 seconds after the power failure, the election procedure of the Tier1-BrokerSet succeeded, and within 5 minutes, the data distribution service returned to normal. In the second test, five IDCs of subordinate points(domains) were selected randomly, of which all network cables were removed. Within 30 seconds after the accident, influenced data subscribers successfully find and connect to the nearest Brokers in Tier2, and within 3 minutes, the data distribution service returned to normal.

Additionally, Tensor has running smoothly online for half a year.

VI. Conclusion

Nowadays, more and more large-scale distributed systems are deployed across IDCs, in order to deal with the latency and reliability challenges of the cross-IDC distribution of critical data, this paper proposes a Redis-based low-latency data publish/subscribe framework. For solving the problem of data loss and duplication caused by network anomalies or cluster node failures, we design a transaction-oriented information transfer mechanism to ensure the eventual consistency of data distribution. To improve the data synchronization performance, we optimized Redis's replication mechanism to make it better suit the unstable network links. What's more, we design an intelligent log analysis based system bottleneck prediction method and a service discovery oriented system failover strategy to ensure the high availability of Tensor. An extensive set of tests on Tensor and the smooth operation of the system in the production environment for half a year prove its efficiency and reliability.

Acknowledgment

This work was supported by National Key R&D Program 2016 (Grant No. 2016YFB0801300); the Strategic Priority Research Program of the Chinese Academy of Sciences (Grant No. XDC02030600).

References

[1] Sathiamoorthy, Maheswaran, et al. ”Xoring elephants: Novel erasure codes for big data.” Proceedings of the VLDB Endowment. Vol. 6. No. 5. VLDB Endowment, 2013.

[2] Redis[OL]. https://redis.io/.

[3] Zhang, Chen, and Xue Liu. ”Hbasemq: A distributed message queuing system on clouds with hbase.” 2013 Proceedings IEEE INFOCOM. IEEE, 2013.

[4] Patel, Dharmit, et al. ”Towards in-order and exactly-once delivery using hierarchical distributed message queues.” 2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing. IEEE, 2014.

[5] Sadooghi, Iman, et al. ”Fabriq: Leveraging distributed hash tables towards distributed publish-subscribe message queues.” 2015 IEEE/ACM 2nd International Symposium on Big Data Computing (BDC). IEEE, 2015.

[6] Yang, Jinsong, et al. ”Data distribution service for industrial automation.” Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies & Factory Automation (ETFA 2012). IEEE, 2012.

[7] CoreDX DDS[OL]. http://www.twinoakscomputing.com/coredx

[8] Apache Kafka[OL]. https://kafka.apache.org/

[9] RocketMQ[OL]. https://rocketmq.apache.org/

[10] Amazon SQS[OL]. https://aws.amazon.com/cn/sqs/

[11] Hernández, Sergio, et al. ”A reliable and scalable service bus based on Amazon SQS.” European Conference on Service-Oriented and Cloud Computing. Springer, Berlin, Heidelberg, 2013.

[12] Liang, Yunlong, et al. ”Study on Service Oriented Real-Time Message Middleware.” 2015 11th International Conference on Semantics, Knowledge and Grids (SKG). IEEE, 2015.

[13] Tencent CMQ [OL]. https://cloud.tencent.com/product/cmq

[14] Dragonfly [OL]. https://github.com/alibaba/Dragonfly

[15] Lu, Z. Y., and Z. B. Guo. ”A method of data synchronization based on message oriented middleware and xml in distributed heterogeneous environments.” 2015 International Conference on Artificial Intelligence and Industrial Engineering. Atlantis Press, 2015.

[16] Kang, Woochul, Krasimira Kapitanova, and Sang Hyuk Son. ”RDDS: A real-time data distribution service for cyber-physical systems.” IEEE Transactions on Industrial Informatics 8.2 (2012): 393-405.

[17] Data Distribution Service(DDS) Version 1.4[EB/OL]. https://www.omg.org/cgi-bin/doc?formal/15-04-11.pdf.