Adaptive Geo–Replication Design

Amadeo Ascó
Trifork Leeds, Leeds, UK

13th November 2014

1 Adaptive Replication

A common technique to reduce latency is the replication of data between different DCs in a system with multiple Data Centres (DCs) distributed around the world, as shown in Figure 1. But keeping multiple replicas is an expensive commodity given the increase in storage requirements and bandwidth as a write operation needs to be propagated to all the other DCs with a replica. The data could be placed in only one DC, in which case there is not any replication, or a copy could exist in all the available DCs, which it is known as full replication. Alternatively the data may be located in some but not necessarily all the DCs such that the number, location and data is determined at run time, which it is known as adaptive replication.

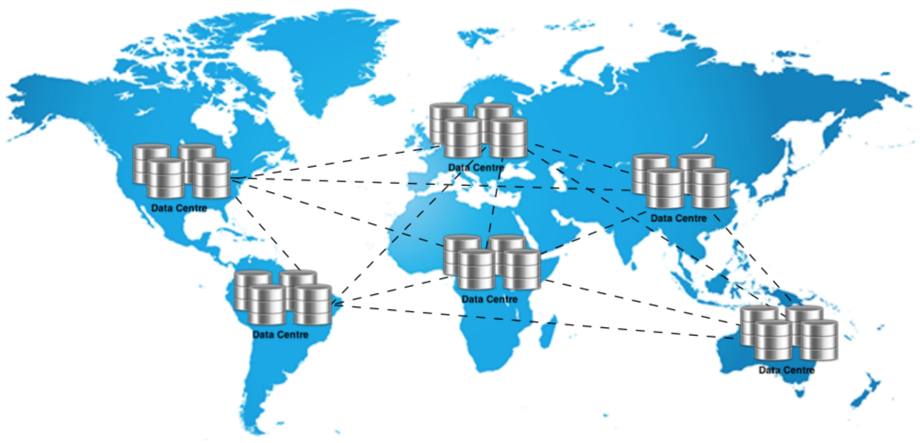

Figure 1: Layers view.

The replications may be grouped, base on its existence, into two types; static replication where a replica persist until it is deleted by a user or its duration expires, and dynamic replication where the creation and deletion of a replica are managed automatically and normally directed by the access pattern of the data used by the users [4]. In static replication the major drawback is their inability to adapt to changes in the access pattern of the data used by the users.

Adaptive geo–replication is concerned in what and where the data is located within the overall system of DCs and how many replicas exist simultaneously, [5, 3, 6, 8, 1, 2, 7]. This is also know as “Adaptive Location of Replicas”. In the example, shown in Figure 2, data reads/writes to DCs 1 and 2 making the data replica to move from DC 3 to DCs 1 and 2 ensuring that the data is closer to where the reads and writes are requested based on the specified objectives and constraints.

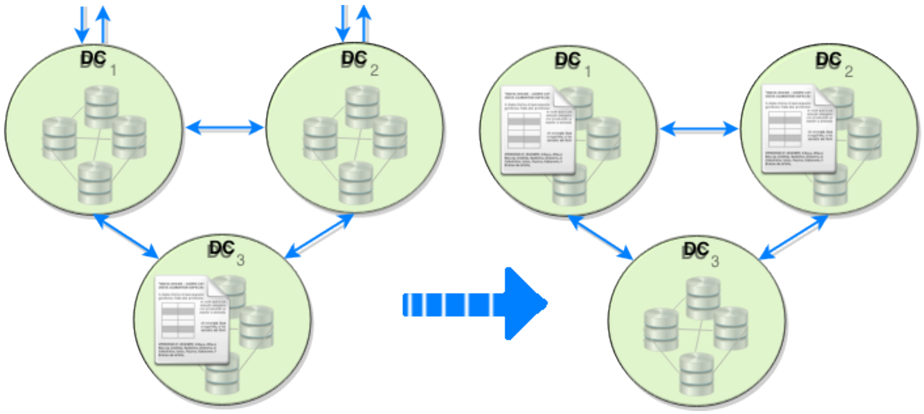

Figure 2: Example of adaptive geo–replication.

2 Implementation

Layers required to dissociate both the strategy and the replication, see Figure 3. These layers are:

- “**Startegy Layer**”
  Responsible to control the replication process.
- “**Replication Layer**”
  Responsible to maintain the replication, deal with the communication between DCs and keep the assurances such as eventual consistency.

All the messages (operations) are received first by the strategy that may convert them to other messages based on its implementation. Some operations are required:

- **Read**
  Request sent by a client to a DC. If the data is not replicated then forward

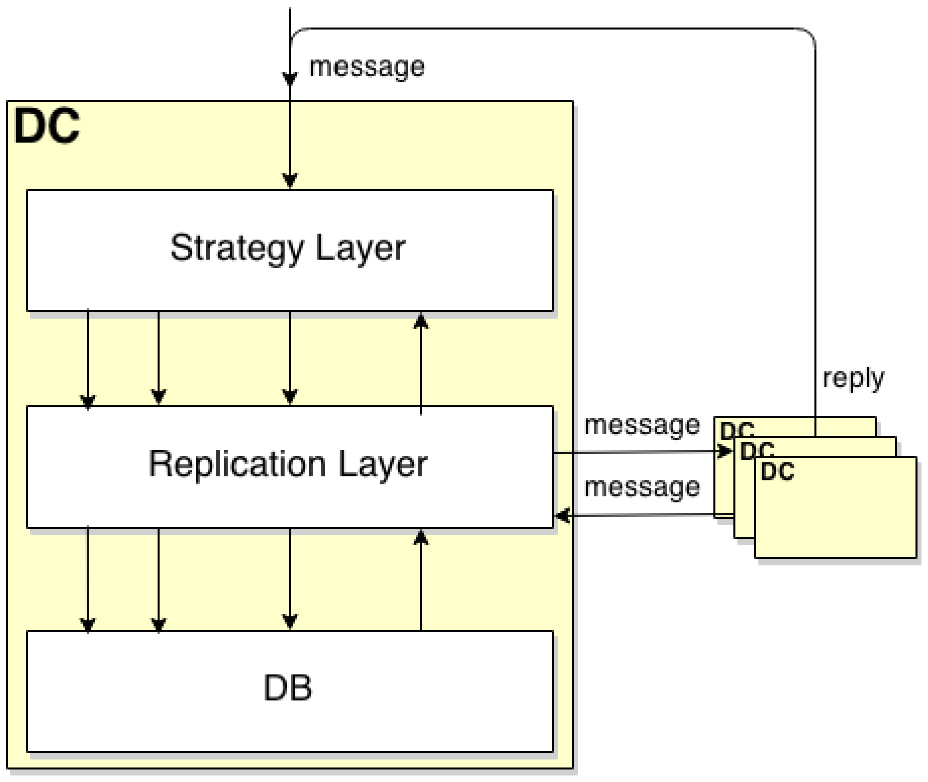

Figure 3: Layers view.

it to appropriate DC, so here it is where “Has replica”messages are sent to all other DCs to determine the most appropriate DC to forward the request, which constitute the discovery state. This may be required to do it once, the first time, and the result may be used in subsequent messages. The DC receiving the forward read message will reply with the data, assuming it does still have a replica, otherwise return a read reply with an error code and the the discovery must be executed again.

It must contain at least a message unique ID, the message type, the originator of the message the data key.

- **Has replica**
  Request sent by a DC to another DC, which will reply only if the destination DC has a replica of the data. This could be similar to a read but without the returning any data, so more efficient if the data to return is large.

  It must contain at least a message unique ID, the message type, the originator of the message and the data key.

- **Create**
  Request sent by a client to a DC to create the data. Once the data is created some other operations may be sent by the strategy from the “Strategy Layer”to fulfil constraints, e.g. minimum number of replicas.

  It must contain at least a message unique ID, the message type, the originator of the message and data tuple key, value.

- **Write**
  Request sent by a client to a DCt to change the value of the data. If the data is not replicated then forward it to appropriate DC, so here it is where “Has replica”messages are sent to all other DCs to determine the most appropriate DC to forward the request, which constitute the discovery state. This may be required to do it once, the first time, and the result may be used in subsequent messages. The DC receiving the forward write message will change the data locally and send update messages to all the other DCs with replicas.

  It must contain at least a message unique ID, the message type, the originator of the message and the data tuple key, value.

- **Update**
  Request sent by a DC to another DC with a replica, which will update its version of the data, the replica. This message type could by implemented as a type of forward message.

  It must contain at least a message unique ID, the message type, the originator of the message and the data tuple key, value.

- **New replica**
  Notification from a DC to one or many of the other DCs to let the DCs know that it has a replica. If the notification is sendt from original DC to one of the DCs with replicas that the destination DC must notify all the other DCs with replicas and reply to the originator DC with a list of the current DCs with replicas.

  It must contain at least a message unique ID, the message type, the originator of the message and the data identifying the new DC with a replica.

- **Remove replica**
  Notification from a DC to all the other DCs that the DC will not have a replica anymore. Here there must be some negotiation between the DCs to make sure some of the constraints are guarantee, i.e. minimum number of DCs.

  It must contain at least a message unique ID, the message type, the originator of the message and the data identifying the DC with a replica, which coincides with the originator of the message.

- **Forward**

  - Read
    Request sent by another DC which it is replied with the loca value of the data (it is an standard read).
  - Write
    Request sent by another DC which changes the local data and sends update messages to all the other DCs.

- **Reply**
  - Read
    The reply message result from a previous read request containing the read data.

    It must contain at least the original message unique ID, the tuple key, value for the data requested.
  - Write
    The reply message result from a previous write request.

    It must contain at least the original message unique ID, and an indication of success.
  - Has replica
    The reply message result from a previous "Has replica"request containing the list of current DCs with a replica of the data.

    It must contain at least the original message unique ID, and the original destination DC or a list of all the current DCs with a replica.
  - ...

All the internal communication could be done by forward messages so requests like "Update", "New replica", "Remove replica"and "Has replica"could be other sub-types of forward messages.

There is also the need to convert replies from other DCs to forward requests into replies to original requesters and send them to those requesters.

It is required a way to get information stored in the underlying "Materializer", such as the list of current DCs.

There is a question about the location of the extra data required by the strategy.

2.1 Messages per layer

All the messages go through the **Strategy Layer** which may convert them to another or many other messages that are passed to the **Replication Layer**. The messages received and processed by each layer are presented below.

At the **Strategy Layer** level the messages are

- From *client*:

  - create
    Create the data for first time. Initially processed in the Strategy Layer and completed in the Replication Layer.
  - read
    Read data with specified key. Initially processed in the Strategy Layer and maybe completed in the Replication Layer.
  - write
    Write specified tuple key, value. Initially processed in the Strategy Layer and maybe completed in the Replication Layer.

- From another *DC*:
  - has_replica
    It is processed in the *StrategyLayer*. Sending this messages to DCs without replica will fail and failure should be ignored, otherwise the DC receiving the message replies with the list of DCs with replicas, inclusive of itself.
  - update
    It implies that a write was done in another DC. Processed in the Strategy Layer and pass to *ReplicationLayer*.
  - rmv
    It indicates that the local replica must be removed, and the data process. Passed to Replication Layer.
  - new_replica
    It is received from another DC that now has a replica. Passed to Replication Layer.
  - rmv_replica
    It is passed to Replication Layer.
  - forward
    - read
      It is processed in the Strategy Layer but may need the Replication Layer, depends on where it is cashed.
    - write
      It is processed in the Strategy Layer but may need the Replication Layer, depends on where it is cashed.
    - ... (own from strategy, potential intercommunication)
      They are processed in the Strategy Layer.

At the **Replication Layer** level the messages all are from the Strategy Layer, which are:

- update
  It implies that a write was done in another DC. Processed in the Replication Layer, so update local value.

- rmv
  The local replica must be removed. Processed in the Replication Layer.
- new_replica
  a new replica exist. Processed in the Replication Layer, so add to list of replicas.
- rmv_replica
  received from another DC that does not have a replica anymore. Processed in the Replication Layer, so remove from list of replicas.
- forward
  - read
    It reads to a DC without replica that it is forwarded to a DC with replica to process it. Processed in the Replication Layer.
  - write
    It writes to a DC without replica that it is forwarded to a DC with replica to process it. Processed in the Replication Layer.

* Needs helper methods to forward messages to all DCs or only those with replicas, something like:
sendAllDcs(Msg) –> to all DCs with or without replica.
sendDC(Msg) –> to a DCs with replicas. This only is valid for when there is a replica in the DC but support could be extended to the case when replica does not exist, in which case, first a has_replica is sent and then the received list is used to forward the message to the DCs with.

3 Acknowledgements

This work is part of the European research project SyncFree.

References

[1] Cristina L. Abad, Yi Lu, and Roy H. Campbell. Dare: Adaptive data replication for efficient cluster scheduling. In *Proceedings of the 2011 IEEE International Conference on Cluster Computing*, CLUSTER '11, pages 159–168, Washington, DC, USA, 2011. IEEE Computer Society. ISBN 978-0-7695-4516-5. doi: 10.1109/CLUSTER.2011.26. URL https://wiki.engr.illinois.edu/download/attachments/194990492/cluster11.pdf.

[2] S. Abdul-Wahid, R. Andonie, J. Lemley, J. Schwing, and J. Widger. Adaptive distributed database replication through colonies of pogo ants. In *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pages 1–8, March 2007. doi: 10.1109/IPDPS.2007.370575. URL http://www.cwu.edu/~andonie/MyPapers/IPDPS%20Long%20Beach%202007%20final.pdf.

[3] Masoud Saeida Ardekani and Douglas B. Terry. A self-configurable geo-replicated cloud storage system. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 367–381, Broomfield, CO, October 2014. USENIX Association. ISBN 978-1-931971-16-4. URL http://blogs.usenix.org/conference/osdi14/technical-sessions/presentation/ardekani.

[4] Xiaohua Dong, Ji Li, Zhongfu Wu, Dacheng Zhang, and Jie Xu. On dynamic replication strategies in data service grids. In *Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on*, pages 155–161, May 2008. doi: 10.1109/ISORC.2008.66.

[5] Junsang Kim; Won Joo Lee; Changho Jeon. A priority based adaptive data replication strategy for hierarchical cluster grids. *International Journal of Multimedia & Ubiquitous Engineering*, 9(6):127–140, 2014. URL http://www.sersc.org/journals/IJMUE/vol9_no6_2014/13.pdf.

[6] R. Kingsy Grace and R. Manimegalai. Dynamic replica placement and selection strategies in data grids- a comprehensive survey. *J. Parallel Distrib. Comput.*, 74(2):2099–2108, February 2013. ISSN 0743-7315. doi: 10.1016/j.jpdc.2013.10.009. URL http://dx.doi.org/10.1016/j.jpdc.2013.10.009.

[7] Thanasis Loukopoulos and Ishfaq Ahmad. Static and adaptive distributed data replication using genetic algorithms. *J. Parallel Distrib. Comput.*, 64(11):1270–1285, November 2004. ISSN 0743-7315. doi: 10.1016/j.jpdc.2004.04.005. URL http://pdf.aminer.org/000/297/337/static_and_adaptive_data_replication_algorithms_for_fast_information_access.pdf.

[8] Zhe Wang, Tao Li, Naixue Xiong, and Yi Pan. A novel dynamic network data replication scheme based on historical access record and proactive deletion. *J. Supercomput.*, 62(1):227–250, October 2012. ISSN 0920-8542. doi: 10.1007/s11227-011-0708-z. URL http://dx.doi.org/10.1007/s11227-011-0708-z.

Glossary

DC Data Centre