HBase Enhancement Proposal

Team5Star

Daniel McVicar - 213027479
Rafay Sheikh - 213033451
Daniel Fortunato - 216796443
Adham El Shafie - 212951018
Yahya Ismail- 213235403

Overview

- Introduction
- Proposed Feature
- Approach 1: Stakeholders, Advantages & Disadvantages
- Approach 2: Stakeholders, Advantages & Disadvantages
- Chosen Approach and Reasoning
- Impact on Architecture
- Risks and Limitations
- Compatibility Testing
- Concurrency
- Issues with new Architecture
- Lessons Learned
- Conclusion

Introduction

- How to enhance HBase?
  - What is HBase incapable of doing *well?*
- Which of the downsides of HBase should we focus on?
- How should this addition be implemented (2 ways)?
- Which of these do we choose? Why?
- How does this affect the rest of HBase' functionality?
  - Testing
  - Diagrams
  - Concurrency
- What issues does this addition introduce?

Proposed Feature - Join Operations in Hbase

Motivation:

- Joins are hard to do in any distributed database – including distributed SQL databases.
- HBase uses query-first design – data that is read together is also stored together
- HBase uses and stores de-normalized data – which works well with join operations

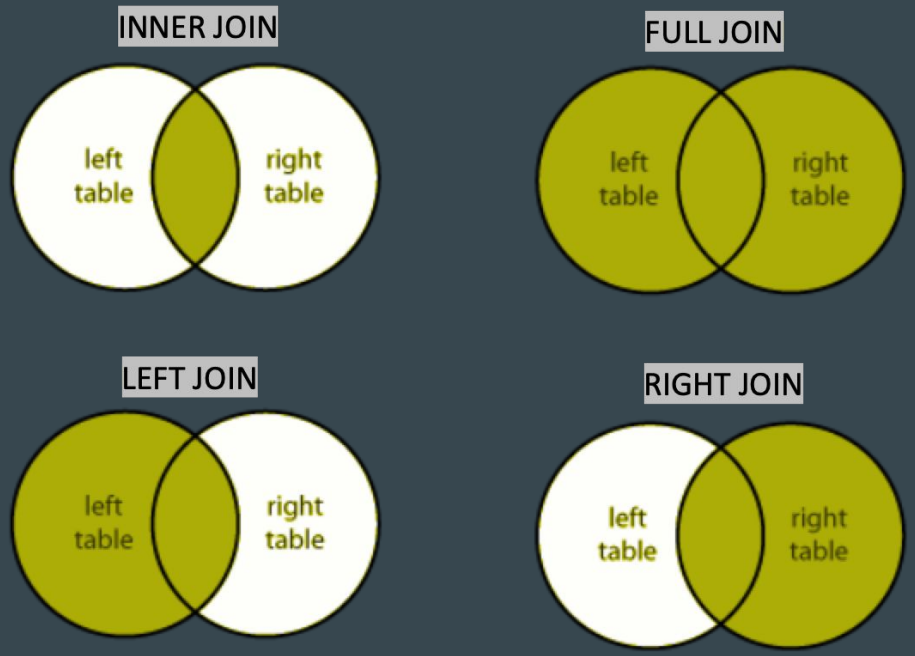
Problems:

- HBase does not have a built in Join operation command for data manipulation
- Alternative methods involve extra setup/configuration of external frameworks
  - Apache Drill, Apache Hive, using MapReduce (not optimized)
- Data has to be imported to other framework and then export back to HBase

Proposed Feature - Join Operations in Hbase

What is a Join Operation:

- operation to combine data from two sets of data (i.e. two tables) with matching columns such that they create a relationship between the two tables.



Stakeholders

- System Users (improve their experience)
  - Individuals don't have to search around for alternatives and waste time setting up frameworks to just get one task done
- Developer (has to maintain, update this feature)
- Clients(small/medium)
  - Who recognize this as a valuable feature
  - Willing to change current system in use to adapt to one with this new feature

Approach 1 - Symbolic Links

Symbolic Links are a Windows Folder management concept used to link folders or files to other folders. This gives the illusion that the folder contains the symbolic file or folder, where in reality they exist separately.

Adapting this idea, we can symbolically link columns and tables to other tables. The advantages of this would be that:

- Data does not have to be rebased (Helpful for large datasets)
- Architecture does not have to be modified a top level file in HMaster takes care of symbolic linking

Approach 1 - Advantages & Disadvantages

Advantages:

- Almost no architecture change
- Very easy to implement solution
- Low performance costs at time of Join operation

Disadvantages:

- Bottlenecked by having to look back and forth in different locations for data
- Editing either table causes the symlink to break
- All calls to lookup or write data double as double the calls must be made

Approach 2 - Join Servers

The JOIN operation can be supported by copying the relevant data onto a separate server. This server can now keep a denormalized (a copy) of all of the data needed for the join operation.

This approach requires a major change to the architecture of HBase. A new “JoinServer”, similar to a region server, is introduced as a slave to HMaster. JoinServer is responsible for receiving copied information from HMaster, manipulating the data as required, and registering a new joined table with HMaster.

Approach 2 - Advantages & Disadvantages

Advantages:

- The denormalized table is fast and easy to search
- Large performance cost to create the join table

Disadvantages:

- All of the relevant data must be copied onto a separate server
- Requires significant architectural changes
- Join table is not automatically updated with recent information and must be recreated occasionally to stay accurate.

Chosen Approach *1* - Reasoning

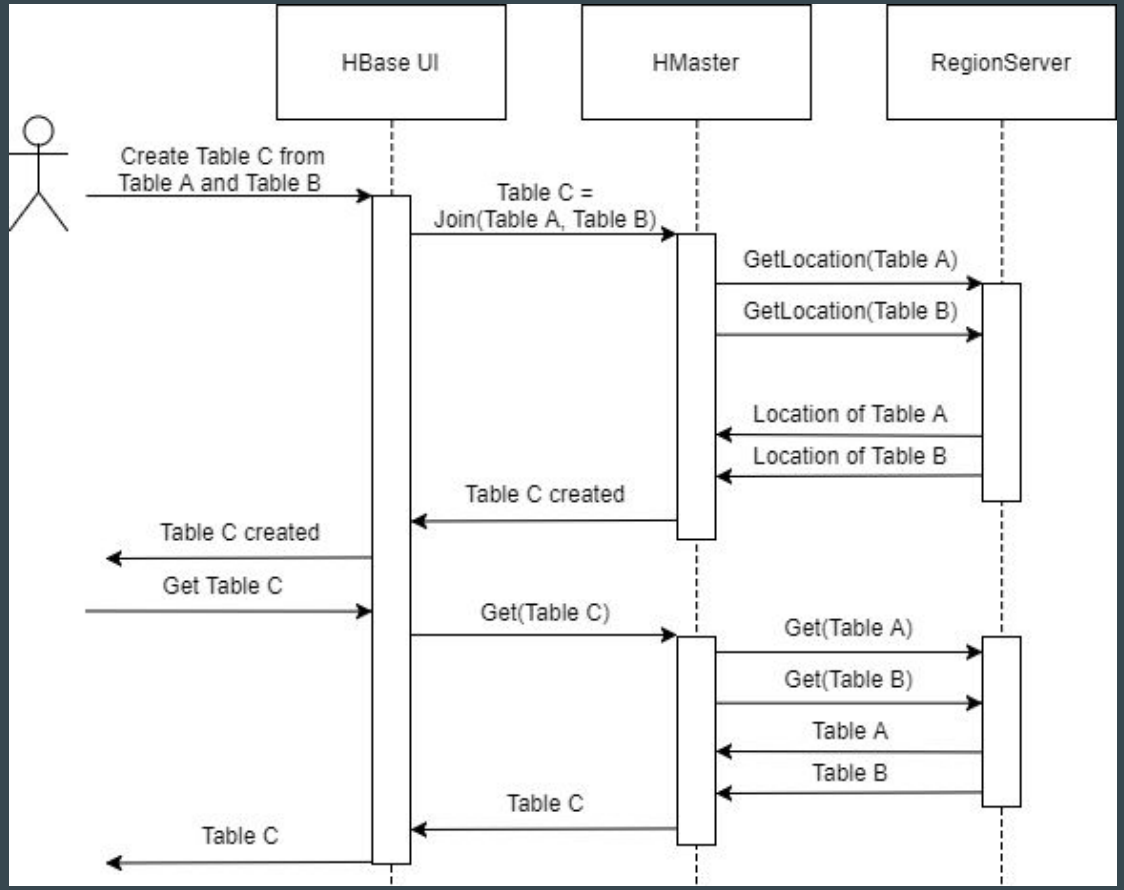
Reasoning	Approach 1	Approach 2
Architectural Change	No major architectural design change	major change in architectural design/pattern
Implementation cost	Low implementation cost	high cost for designated server/cluster
Performance (depends on size)	Faster join operation but longer read/write	Fast for many reads and not many writes

Impact on Architecture

With the SymLink approach there is little impact to the overall architecture

- A new file (Called SymLink Directory) is created in HMaster, this file holds all current links between columns and tables
- The read operation on Joined data must now be replicated to both datasets before returning a result
- The write operation on Joined data must replicate as well as put a write lock on both columns in both datasets (for concurrency)

Use case


HBase UI
HMaster
RegionServer
Create Table C from Table A and Table B
Table C = Join(Table A, Table B)
GetLocation(Table A)
GetLocation(Table B)
Location of Table A
Location of Table B
Table C created
Table C created
Get Table C
Get(Table C)
Get(Table A)
Get(Table B)
Table A
Table B
Table C
Table C

Risks and Limitations

- Additional reads/writes operations are needed on joined data
- Additional locks are needed for write operations
- Possible bottleneck situation when looking for data
- Any change to the table breaks the SymLink
- Runtime of MapReduce might be affected since there are changes in the logic of the placement of data

Compatibility Testing

Key metrics for a database system include:

- Write speed
- Read speed
- Effect of scale on write and read speed.

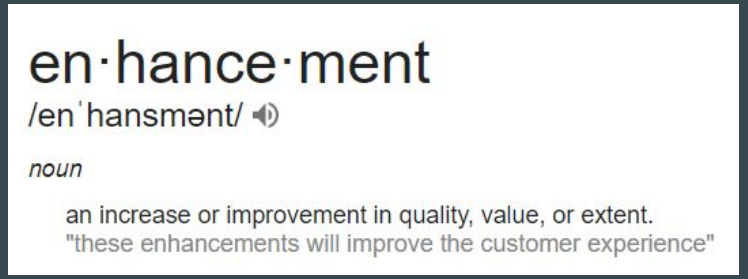
To test the system an unmodified HBase database will be compared against an HBase database with symbolic link joins. The difference in read, write, and scalability can then be compared and the value of the feature evaluated.

Concurrency

With the addition of symbolic link joining HBase will support read concurrency without any additional work.

Write concurrency requires all of the symbolically linked data rows to be locked. Once this is done the write operation will operate concurrently.

Lessons Learned



- HBase, like many other distributed databases, lack the *join* functionality.
- The best approach to introduce a new feature (HBase or any normal code) is the least intrusive one
- Joining via MapReduce is not optimal
- 3rd party applications (Apache Spark/Hive etc.) support their own version of *join* but are not the most efficient for HBase (Big Data Table)
- More efficient to have HBase implement and handle its own *join* operation that can be called on by the client

Conclusion

- Joining could only be done by 3rd party applications, and even then it was inefficient
- A built-in implementation of *join* allows HBase to have the most efficient joining operation that can be integrated in it
- Multiple approaches allows for a wider perspective on how HBase handles different implementations
- After analysis, SymLinks is decided to be the least intrusive approach
- Saves time on join operations, but not the most efficient in lookup time and may need to be called on again should the tables be edited in any way