Storage Systems
Dr. K. Gopinath
Department of Computer Science and Engineering
Indian Institute of Science, Bangalore

Highly scalable Distributed Filesystems
Lecture - 42
Highly scalable Distributed Filesystems_Part 4: Problems in the GFS design model & solutions, BigTable

Welcome again to the NPTEL course on Storage Systems. So, in the previous class, we were looking at the global file system sorry the Google file system GFS. And today we will look at some of the issues the design throws up and see what can be done about it.

(Refer Slide Time: 00:55)

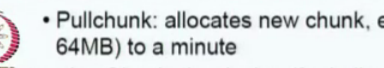
Summary

- First really large FS
  - Reliable
  - Scalable
  - Available
- No POSIX...

So, as I summarized last time, this is one of the largest file systems pretty still today I think it may be the largest. So, of course, they have moved to a (Refer Time: 01:07) different version of this GFS, sometimes that is called GFS 2, sometimes called (Refer Time: 01:14) certain about, the various names are there. So, still the living on in a slight different form and as we discussed before it is not POSIX. And one critical thing about how without POSIX, it is still ok is because they have ensured the applications are designed. And the way they do it is by using additional infrastructure on top of this; and the infrastructure one important infrastructure place on top of this GFS is something called the BigTable.

We will look that at the thing a bit you will not go it too much integrate because it is closer to application level ideas, but we will take a quick look at it to see what all it does and how it fits with the general GFS model. Before that we will take a look at what are the problems with GFS.

(Refer Slide Time: 02:08)



So, what is the issue with GFS? First of all the design says high sustained bandwidth more important than low latency, and basically design for batch oriented applications such as web crawling indexing. So, these are the critical need, and the early 200 or late 1990s, but now what is happened is that Google has come up with other applications like Gmail or YouTube etcetera, these are closer to real time. Especially this Gmail is a just like user interaction it absolutely critical that there has to be some low latency also is an important issue. So, if you try to when Gmail or YouTube on top of GFS you might get into problems. We will see what kind of problems are there.

First of all because of these kind of near applications the developer base has moved from a batch oriented frameworks like map reduce to interactive applications. So, because you know other doing batch processing it turns out the original design was not a disaster. What is the original design, which had a single master, and there are single point of failure SPOF, but it is unacceptable for latency sensitive applications such as video server. In the beginning, in early iterations of the GFS, there was no automatic failover scenario if master crashes in the very, very beginning and should be manually restarted it

go to as much as 1 hour, because you have to look at the logs and replace the log etcetera, required quite bit of time.

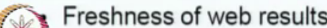
Now, there is supposedly I am doing all the detail, but this is what I read, there are supposed to be in automatic failure, but takes 10 minutes, but now it is closer to 10 seconds, but even this 10 seconds is a problem because especially if we are talking about real time situations. So, as mentioned there are shadow masters, but even those take some time to come up online. So, basically also what happens is that you have to write the 3 chunk servers, there are model of consistency is that model of completion of a write request is write to 3 chunk servers. And then if one of them happens to be slow, it is slow then everybody because you wait for all three of them to respond before you proceed. So, this limit set to somewhat lower bandwidth situations. And basically you want to take allocating a new chunk etcetera and then writing all those things essentially takes about multiple seconds. So, this is not a really good design for low latency.

Other issue about this GFS is that because it was expected to be a batch processing application, they do a lot of Qing in the design. So, Qing is a way to smooth out variance in terms of arrival of Java's etcetera. So, Qing in the design helps you to smoothening the flow of activity in the system, but the minute you start Qing you have to first enter the queue and exit the queue that itself will is there is some kind of delay that comes into picture. And basically the Qing delay essentially kills this low latency possibilities. So, Qing is good for throughput, but generally bad for latency, so that is in a way it turns out to be somewhat problematic serious problem.

(Refer Slide Time: 05:55)

More Probs

- Limited number of files (all metadata in master in memory!)
  - 64MB to 1MB?
  - OK upto multiple 10's of TB.
  - With 10-100's of PB? 10GB-100's GB of mem.
- Single master a scanning bottleneck (recovery, etc)
- Metadata serving bottleneck with many Map/Reduce clients
  - Scaling from 1000's of ops to 10000's and beyond
- Batch op does not allow for large-scale incremental processing using distributed transactions and notif cations
  - Freshness of web results



Other issue that is an important issue also for GFS is that all the metadata in the master is in memory; that means, it can only handle limited number of files, because in each file has about 100 bytes or so of data that will essentially limit the number of files. So, if you really want to go from 64 megabyte chunk to 1 megabyte chunk which is also required for other reasons then this becomes the serious problem. So, in the beginning that design was 10s of terabytes or more, but now it is 100 of petabytes I am told that it is certainly crossed 100-peta byte.

So, with 100 of petabytes, the amount of memory required for the master has this crossing about 100 gigabytes plus. So, this also is a initial. So, if you have a single master you do not have parallelism because you cannot if there is doing some scanning recover etcetera this single master turns out to be a problem also same thing within a multiple map reduce clients you will find that metadata serving also becomes a problem. So, the early generation of GFS both do 1000 operations, but we will want to do at least one or two (Refer Time: 07:32) higher and so having a single master is a problem.

So, having shadow masters like was designed earlier as we discussed earlier is of some help, but as we discussed before the shadow masters are slightly behind with respect to updates of the master. So, other issues that nowadays there is an important requirement that if you have a new page you design and put it up on the web you want people able to see it fairly quickly. Now, batch operations does not allow for large scale incremental

processing using distributed transaction notifications. Basically what happens is that you are limited to you are batch processing which can take about a day also. And therefore, the freshness of web results also becomes a problematic.

So, what you really need is every time a new page comes online, you want to able to somehow get some triggers that something new has come in. And using the trigger you want to able to do some incremental work and put it up on to your this one, so that we have freshness of web results, but having this batch kind of model does not really help. So, the batch is basically a killer for this kind of issues.

So, nowadays Google has gone has attempted to a address this problem systematically. And now told that if you put up a web page it can be available to you within 10s of seconds or 100s of seconds I do not know what number is they say that it can be made available as a result in the web search. Precisely because they can do incremental processing essentially they have some kind of triggers which tell you that something has been added to it and they immediately go on call it and index it immediately. So, instead of multiple hours, it is now closer to let say minutes or tens of minutes or sometimes you say event say tens of seconds which are remarkable thing which is not possible batch processing.

(Refer Slide Time: 09:50)

So, we have to do something else; other issue is we already looked at the consistency issues. So, I think we have gone through some of these aspects, we know we see that the

clients keep on pushing right until it succeeds. Because we have to write to three places some other can fail, we declare that depending one of them fails use redo start from the beginning keep on pushing until happens, but if you have client failures then it causes indefinite state because there is no client to push the right.

So, again similarly there is record append interface I think we discussed it for multiple writers you want to write to a log concurrently, but what can happen is that the primary itself can change as the write is going on because of failures. So, primary selects offset to write primary can change and new offset is taken. Therefore, record append does not offer any replay protection, some piece of data multiple times in the file or data in a different order depending on how the new primary schedules the writes because it the new primary decides how the various concurrent write operations are going on. So, therefore, it is possible to have the data in multiple times in one chunk replica, but not in all on all.

So, for read out data, data can be available in different ways at different times. So, application has to work around this thing which is the big issue something at record level records in different orders depending on which chunks are read. So, there are some other issues originally it was told that the all this things can be handled somehow that has previous standard applications newer applications this term not to be a issue big issues that had to be resolved somehow.

(Refer Slide Time: 11:52)

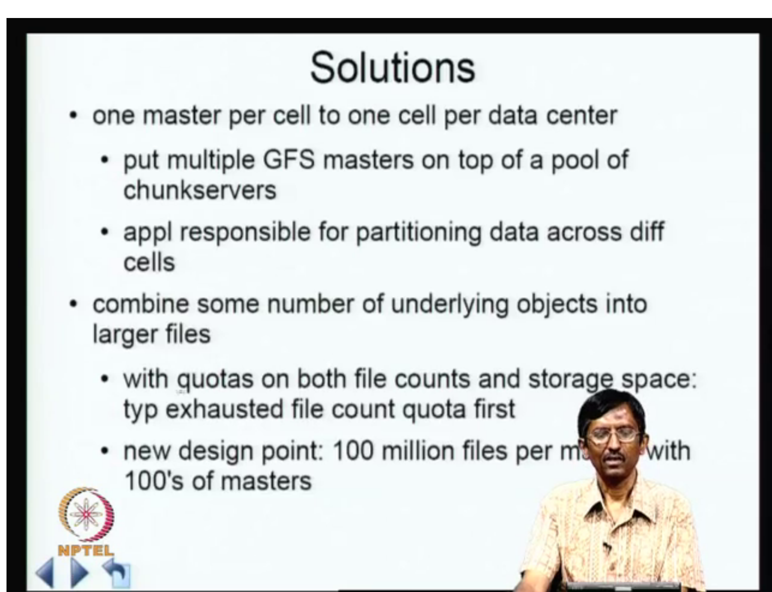
Other probs

- snapshot a chunk: to replace a replica, or whenever some chunkserver goes down and need to replace some of its files
    - Actually cloning
    - Difficult...

And the system also had some other issues like snap shooting which is a long running mistakes a long time. So, snapshot being generally is a very difficult operation as we noticed we have to take back the leases you have to wait for all the leases to come back all kind of (Refer Time: 12:12) it takes time.

So, generally a snapshot in any file system is difficult and they had provide this functionality, so that you can replace the replica or whenever chunk server goes down we need to replace it files basically have a snapshots where you can repopulate it very quickly. But this turns out to be quiet complex and this has slowed down or created difficulties in other parts of the system.

(Refer Slide Time: 12:40)



So, what are the solutions with this kind of problems, basically you have multiple masters and each master is it should be one master per this let us enter that is why. So, a basically a put multiple GFS masters on top of a pool of chunkservers. And you try to still avoid the complicated consistency issues that crop of because of this and the way you do it is by partitioning data across different cells. To make sure that each cell actually has data; which is not common with other cells. So, there is no issue of trying to do consistency across these cells. So, the other issue is, is it possible to combine smaller objects and put it into bigger chunks, but people are discovered that if you count the number of there are some limits in the system, I think as I mentioned number of files has a impact on the size of metadata that the master has to keep.

So, they tried both quotas on file counts and storage space itself and typically their experience has been that you typically exhaust the file count quota first that is you still have a lot of space, but you do not have enough space to keep the metadata in the memory you know of the master. So, they have come with a newer design point 100 million files per master with 100s of masters because these are kind of new design things are starting to. I think already is available on both I do not know that exactly does some other details are not available right now exactly what they have done. I think the new system called (Refer Time: 14:30) probably have all these issues as all these things taken care of.

I picked up some of these material from various interviews given to various top designers of Google file system they discuss what the problems are and, but there is no paper as far as I know. Let us talk about how they are solved a problems, complete paper like the Google file system paper which came in a (Refer Time: 14:52) as to not that level of comprehensive detail. And consistently discuss all the issues that is one thing that there are other issues. So, these are the previous solutions for at the level of the systems level.

(Refer Slide Time: 15:11)

Appl Workarounds

- Gmail uses a multihomed model: if one instance of a Gmail account not avlbl, move to another data center
- BigTable transaction log a big bottleneck for getting a txn logged
  - two logs open at any one time
  - write to one and if that gets stuck, write to other
  - merge logs but only on replay
- or, use BigTable for any app with lots of small data items

We can also have application level work around. So, Gmail uses what they call a multihomed model. So, there are multiple essentially instances of your Gmail account. If one instance of a Gmail account not available, you just move to another one and then

when you think about reconsidering its later from availability point of view, so that is one. So, an application essentially starts out assuming that it is a multihomed model and there is no systems the systems infrastructure is not giving you a let us say a uniform view without worry about any details. You have to think take about yourself.

Other issues for example, if you take BigTable as an application on top of the Google file system it has got a transaction log and say this application log you might call it, it turns out it is a big bottleneck because everything has to go through it. So, what they have done is they have decided to duplicate the logs there are two logs open at any point in time you tried to write to one and because it is writing to three replicas and all those things if it gets stuck. If it signs turns out to be slightly slower than expected you just start move to the other one; so things of that kind.

But then things will go out of work. So, the logs will be slightly not exactly consistent it is with respect to each other. So, you have to merge it, but a good thing is that you do not have data all the time you do it only when you have to some failures taking place in every plate at that time you figure out what to do. So, that is another way to work around the problem. So, the other issues of this kind that have corrupted when various people have tried to do various different things. And the other one other approach is to use the completely newer type of infrastructure on top of GFS which can be used to handle many of these issues and that particular model is called a big BigTable. It is for any application with lots of small data items, so that this BigTable allows you to create items delete items in memory and slowly it is pushed to disc and its bigger chunks.

So, in the sense this gives the ability to aggregate small piece of data and then do clean that particular data at regular intervals compact them and then later when you want to make it persistent you can push it out to global file system the big blocks this kind of things can be done. So, the idea is to do most of the stuff in memory and then get the updates and then clean the updates that is whatever is needs to be garbage collected whatever updates are no longer necessary throw them out keep on compacting them. But reduce the amount of memory if possible when finally it becomes bigger than what you can handle, then finally start thinking of pushing it out to disc, and finally, GFS is the is now there in a picture GFS is essentially storing it persistently.

(Refer Slide Time: 18:39)

BigTable

- a structured storage system with many key-value pairs and a schema
  - For web indexing, Google Earth, etc.
- sparse, distributed, persistent multi-dimensional sorted map
  - map indexed by a row key (upto 64KB, typ 10-100B), column key, and a timestamp: each value in map an uninterpreted array of bytes

    (row:string, column:string, time:int64) → string

    - Eg: URLs: row keys, metadata categories of web pages: column names, contents of web pages: contents: column, timestamps when fetched
  - data in lexicographic order by row key
- every read or write of data under a single row key is atomic (regardless of the number of different columns being read or written in the row)

So, let us just quickly look at BigTable, is a structured storage system with many key value pairs and a schema. So, this is for web indexing or things like Google earth etcetera. Essentially to putting it is like a different way it is a sparse distributed persistent multi-dimensional sorted map. So, essentially this is what is often called the key value store. We have a key and then there is a you give the key and look it up, look at the table. It is persistent that means, that once you collect enough data of these tables, you can make it persistent; and it finally, is going to be stored by the Google file system. It is distributed basically because it is the various key value pairs will be stored across various nodes across the system. This is multi-dimensional, because it can have essentially it is this value can you many components, you can call it multiple columns.

So, you can take the key as the representing the one row and the value being multiple columns. So, essentially it is like a table just like a table with many, many rows and many, many columns. And each this particular map is indexed by row key; and a column key and a time stamp. Basically the timestamp is used, so that you can have multiple versions of the same piece of information, and you always keep the most recent information accessible the earliest. So, later once you have to take some trouble to travel list.

And to make it as general responsible each value in map in an interpreted array of bytes just like for example, like the way UNIX also treats files it is uninterrupted sort of bytes

same thing same here anything, you can put a that is not structure there. Only structure you have is that it is a sorted map and it is sparse distributed multidimensional persistent structure. So, for example, so what BigTable is essentially you can summarize as follows is a row, is a string, column, which is multiple is a string, but it can also be seen as multiple strings, so that you have multidimensional aspect rate. Time is basically set of let say time instance at which these particular data were created or modified. And its int64, so if you want to it is your business to make sure that this is different from every other time instant. So, we have to ensure that this real if n is real time, but if real to have must to be have a same time for example, then you have to find some way of changing it.

So, an example of the way BigTable can be used is you can store, for example, if you are crawling the system crawling the work then URLs could be row keys various metadata categories of web pages column names. For example, you may want to categorize the web pages as depending on which language it is for example, whether it is English or any other language Hindi or whatever right, so that is basically the column names. And then we can also have contents of web pages that also is another column, and in that time stamps.

So, and all the data is usually kept in lexicographic order by row key, every read or write of data under a single row key is atomic in spite of the number of columns etcetera. So, one thing that you guarantee is that reading writing of data under a single atomic row is atomic they do not say anything about what happens across rows, but the single row is what they guarantee. And you have to figure out, the application has figure out how to get their job done with this particular guarantee.

(Refer Slide Time: 23:30)

BigTable Design

- Not a full relational data model
  - provides clients with a simple data model that supports dynamic control over data layout and format
  - allows clients to reason about the locality properties of the data represented in the underlying storage
    - Clients can control locality of their data through careful choices in their schemas
- Data indexed using row and column names that can be arbitrary strings
- Treats data as uninterpreted strings, although clients often serialize various forms of structured and semi-structured data into these strings
- Bigtable schema parameters let clients dynamically control whether to serve data out of memory or from disk

A first symbolic thing that is this BigTable is essentially a storage design, it is not a database design, it is not a relational data model. Basically the understanding is that databases have problems on scalabilities serious problems on scalability they cannot cope up to this kind the kinds of demands that current web address places. So, the idea is not to go with the relational data model and it provides client is a simple data model, but supports dynamic control over data layout and format, so and the way that is done it by what we just discussed already. We have a set of rows set of columns and you just have a key value pair kind of model that is basically a simple data model.

And at allows clients to reason about the locality properties of storage represented in the underlying storage. So, essentially they can the line clients can dynamically control whether to serve data out of memory or disc, all these things are possible clients can control locality of the data through careful choices in their schemas, they are provide enough controls, so that the clients can figure out the issues with respect to locality. So, the data can be data indexed using row and column names that can be arbitrary strings. Again as we said before it treats data as uninterrupted strings, but clients can serialize it into various forms of structured and semi structured data onto these strings.

Basically in some sense you decide how you serialize the data as a string that is up to you but this is. So, things similar again to the way the UNIX file system level when using an file it is your business how to make it as suppose series of bytes and summarize

your control. So, it is essentially a some kind of UNIX kind of model.

(Refer Slide Time: 25:46)

Design (contd)

- Row range for a table dynamically partitioned.
  - Each row range a *tablet:* unit of distribution and load balancing
  - reads of short row ranges efficient: typically require comm with only a small number of machines
- Clients can select their row keys for good locality of data accesses
  - For storing Web pages, pages in the same domain grouped together into contiguous rows by reversing the hostname components of the URLs

Row range for a table dynamically partitioned. So, you can take as I mentioned all the data is lexicographically ordered, and say basically you can take a set of rows and then you can dynamically call it one tablet. And this tablet is unit of distributional load balance, multiple rows become a tablet and that is basically unit of distribution load balance. Because it is that way reads of short row ranges addition basically they all are there is a single machine. So, it requires if it is a fairly large tablet, it might at the most span of few machines. So, you just tried to talk to a few machines.

Clients can select the row keys for good locality of data access. Again client is given complete control about how they can order things. For example, if you are storing web pages the way you can do it is to get locality is to reverse the URL. If you reverse the URL the last component is basically the site name. And therefore, you can use that as basically pages in the same domain grouped together. So, example if your website is x dot y dot z, you make the name as z dot y dot x and z dot basically is you might call the major let say organizational boundary, and therefore, all those pages will be connected to problem basic similar in the same place. So, pages in the same domain grouped together by reversing hostname components of the URLs.

So, again this is all up to you this is nobody is telling you that is data has to be done a clients can design all these things.

(Refer Slide Time: 28:00)

Column Families

- Column keys grouped into sets (column families): *family:qualifier*
  - basic unit of access control
  - All data stored in a column family typ same type (eg. language)
    - compress data in the same column family together
    - number of distinct column families small (100's) but a table may have an unbounded number of columns
    - families rarely change during operation

So, other thing that they have is what they call com column families, column keys grouped into sets, and basically this is the family and the qualifier for example, family could be language file. And good think about this is that all data stored in a column family typically same type, and probably they might even have same type of data also. So, it turns out because of this, you can compress data in the same column, you should compress them in the same column family together you get much better compression ratios. For example, in one column you may have some names; it is very likely that there is a lot of commonality in the names.

So, if you compress a group of columns in a column family there is going to be lot of possibilities for compression (Refer Time: 29:00) across rows usually the rows will have columns from desperate types of objects. And therefore, they will not probably produce at all, so that is also one of the reason. So, the number of distinct column family is small, but a table may have unbounded number of columns right number of columns may quite large, but column families that will be reasonably small.

(Refer Slide Time: 29:29)

BigTable, GFS and Chubby

- GFS provides only two basic data structures: logs and SSTables (Sorted String Tables)
  - majority of data in protocol buffers (data description lang) in these two structures.
  - SSTables are immutable, while BigTable provides mutable key value storage
  - Stores incoming data into transaction log files.
  - "compacted" into a series of SSTables, which in turn get compacted together over time (~LFS)
    - Sort and index
- Uses a highly-available and persistent distributed lock service called Chubby that uses Paxos
  - 5 active replicas, with one elected master to actively serve requests

Again BigTable it is so other infrastructure as we already looked at, as we already seen it is of definitely GFS. It is also something as called chubby. First thing to notice is that GFS provides only two basic data structures logs and what they call SS tables. And they also have something called protocol buffers, you are not really going to (Refer Time: 29:58). But basically the idea is that if there is some understanding about how to specify how data structured then there is going to be less afforded transforming one type of data type into some other type data base are required. Often time this take a lot of computation memory and energy. And for example, data format it could be one form in java, it could be some other form in some other language and every time you move this kind of boundaries application boundaries or system boundaries, may have to convert it from one type of format to other type of format.

Once it start doing it systematically, the inefficiencies start mounting seriously and you will find that there is going to be lot of performance loss. So, one thing they have tried doing also is have at application levels some idea about how you structure the data and the name that given is something called protocol buffers; essentially, in saturation we call it data description language. It tells you how the data should be described possibly on things like how it is different systems will have different types of padding or other things in spite of all these things you should be accessed in a uniformly same way, and avoid as much as possible the conversions from one format to another format.

So, basically what happens is that the you are transferring data from one system to another system using this kind of protocol buffers, and these things are finally, ending up in logs or what is called SS tables. And these things are persistent, but while these things are being persistent, you need to keep them in memory as are being constructed, so that is where the BigTable and GFS come to picture. The SS tables are immutable that is once you have figured out that you want to store that amount of a logs that have been created because of changes in the system, once they cross a certain boundary and it can be easily return to one chunk for example, then they become immutable.

Whereas BigTable is working on in memory and it will give you the storage that can be changed because you can add a row delete a row you can remove the column add a column all kinds can be do. So, all those things operations will be done in memory and once all the changes logs etcetera corresponding to these operations, you want to make them persist, then the finally going to reside in the GFS kind all GFS.

So, again if you think about the divisional labor, you see that GFS provides this something called SS tables this are immutable means are persistent and either you read them completely into memory change it in the right hand. Essentially you might say its equivalent to doing read modify or write operations and that is what they call it immutable we cannot really change it there is no APA by which you can go and say into a chunk change this part. We have to get into memory and do something with a done or write it back fully, while BigTable provides mutable key value storage.

So, essentially what is happening is that any incoming data is stored into transaction log files; and then as you proceed in the system, some of those changes might may no they might have to be thrown out, you can essentially say that some of the changes are let us say absolute you can throw them out. So, basically we can compact them again and then you keep on doing it finally, once you find that you are running out of memory etcetera. You want to store it persistently then you push it out to GFS. They similar to a log structure file system in a log structure file system, what you do is you there is no home locations like in a general kind of file system, basically all the operations on the file system happen in memory as a log.

And as a log keeps increasing all the changes are actually stored in memory called segments. And once the segment becomes full you want to reuse that memory for newer

types of changes then you what you do is you clean them you clean those segments, so that all the absolute changes that no longer needs to be kept around for example, you modified a file three times. So, intermediate changes need not be stored they can be thrown out you just try to keep the last one. So, you can clean them and then write back to disc. And the idea there of course, is that once you accumulate a megabyte or few megabyte kind of changes logs, you can write it in efficiently, you can write those long bigger chunks of writes (Refer Time: 35:13) that is what log structure file system does. So, here also it is the same thing.

So, the BigTable provides immutable key value storage you accumulate the changes and the form of logs. And then the logs something become, you clean them now and then, so that you can memory more effectively. But once you have come to a stage where in spite of cleaning etcetera, it is going beyond the size that you want to keep in memory then you start pushing it out to disc. And the way you do it is to send it out as chunks to GFS that is how GFS and BigTable work together.

Other thing that the system uses is something called chubby which is basically a highly available and persistent distributed lock service and uses Paxos. So, it is got five active replicas with one to elected master to actively serve request. Again this is lock service, so it is used in whenever you want have mutual exclusion across a multiple, for example, you want to make sure this only a single master then use a lock serve the Chubby.

(Refer Slide Time: 36:35)

API

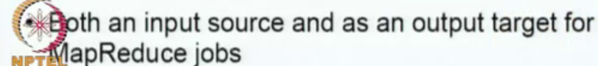
- Creating and deleting tables and column families
- Changing cluster, table, and column family metadata, such as access control rights
- Client applications can
  - write or delete values in Bigtable
  - look up values from individual rows
  - iterate over a subset of the data in a table

So, what are the things that BigTable provides, it creates you can create and delete tables and column families. You can change the some metadata for example, or corresponding to some of these objects like tables and column families. You can the client applications can write or delete values in BigTable. Look up values from individual rows, or iterate over a subset of the data in a table, you can do all these things.

(Refer Slide Time: 37:12)

Use cases

- Supports single-row transactions to perform atomic RMW sequences on data stored under a single row key
  - No support for general transactions across row keys
  - Bigtable allows cells to be used as integer counters
- Supports execution of client-supplied scripts in addr spaces of servers
  - Allows data transformation, filtering based on arbitrary exprs, and summarization via operators but no write back into BigTable



And various ways we use the system is that single row transactions to perform atomic read modified write sequence under a single row key is possible, basically get the whole row change whatever you want to change and put it back. Because you have single row transactions to atomic with respect to both read and write, but you do not have any general transactions across row keys, multiple row keys not possible. They also has some additional features, basically they are able to do read only memory maps what I can understand therefore, you can support execution of clients supplied scripts in address space of servers. So, allows data transformation filtering based on arbitrary expressions, and summarization via operators, but no write back into BigTable.

So, I think from what I can gather from this is the daily some kind of memory map is available on the intermediate memory structures. And they can be used as a way to do everything in memory, but you cannot write to BigTable you still have to go through the sequence as I mentioned earlier that is basically this sequence go through SS tables and then make it persistent, so that is the only way to write it back to BigTable is to GFS. So,

the idea again is to make sure that you can efficiently execute various types of operations like data transformations of filtering in memory. And I think from what I gather it is going to be its using memory map to n map to do all these things. Other thing that they support is they have some various facilities, so that map reduce jobs can be supported back basically this BigTable can both the input source as well output target for map reduce jobs.

(Refer Slide Time: 39:34)

BigTable Impl

- 3 main components: a library that is linked into every client, one master server, and many tablet servers
- tablet servers dynamically added (or removed) from a cluster to accomodate changes in workloads.
- master responsible for
  - assigning tablets to tablet servers
  - detecting addition and expiration of tablet servers
  - balancing tablet-server load
  - garbage collection of files in GFS.
  - also handles schema changes such as table and column family creations
- persistent state of a tablet stored in GFS thru memtable

So, what are the various components of a BigTable, they have a three main components a library that is linked into every client again they have a single master server and many tablet servers. So, tablet servers dynamically added or removed from a cluster to accommodate changes in workloads. So, basically what is a tablet we mentioned that a tablet is bunch of rows. So, we can have various machines, various nodes will have various tablet us, and therefore, that is a way you can get the scaling that you as to increase the performance.

So, the tablet servers can be dynamically added or removed from a cluster to accommodate changes in workloads. So, this have the dynamicity in a system is provided, and the master is responsible for assigning tablet us to tablet servers. So, tablets are basically bunch of rows and tablets servers are nodes so somebody has to do assignment and that is what the master is doing. Again there the reason why this gone for single master is for the same reason why GFS has gone to a single master for the same

reasons, so basically that the single masters complete view about what is going on.

So, again detecting addition and deletion of tablet servers, basically tablet servers dies or comes online we can do these things, balancing tablet-server load, so again you can move tablet us around or resize the tablet us depending on the load. Again even garbage collection of files in GFS is initiated by a BigTable master. So, if there is an extreme if there is serious short in spaces etcetera some of them can be made to start doing up garbage collection immediately. So, then other things that the maser does is also handles schema changes such as table and column family creations we have to add a co column family then you have to essentially do it for all the rows right. So, you have to do something you have to do systematically go iterate over all the rows and do something with it. So, all this kind of things are handled through the master. So, again as I mentioned the persistent state of a tablet is stored in GFS through MEM table.

So, basically MEM table is in memory structure which has all the changes I think like we discussed here before. So, all the changes for example, they go in to the MEM table and then this MEM tables are compacted and these SS tables are created and these SS tables get compacted together over time and they finally, get in to GFS. So, that is what persistent state of a tablet is stored in GFS through operations on the memory table.

So, I think I am end of my discussion today. I thought I will take more time, but this is where I am. So, I think I will continue next time with further discussions on BigTable.