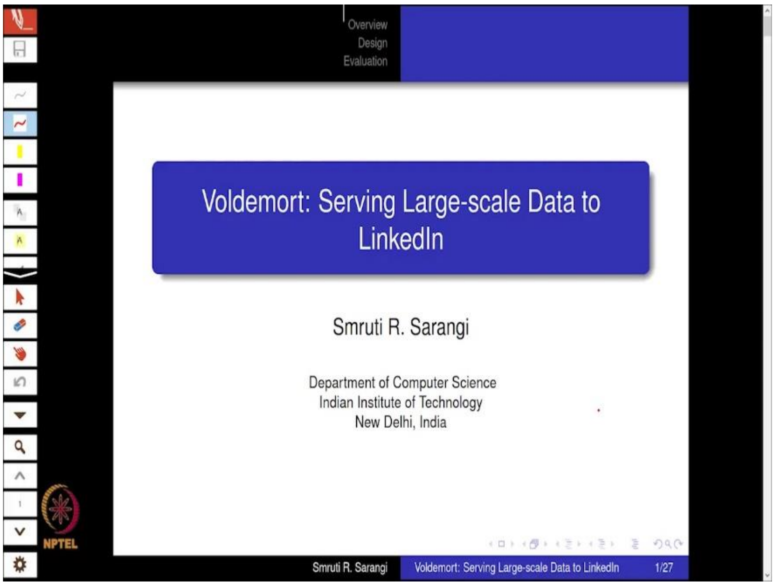
Advanced Distributed Systems
Professor Smruti R. Sarangi
Department of Computer Science and Engineering
Indian Institute of Technology, Delhi
Lecture 23
Voldemort: Distributed Storage Engine for LinkedIn

(Refer Slide Time: 0:17)



Overview
Design
Evaluation

Voldemort: Serving Large-scale Data to LinkedIn

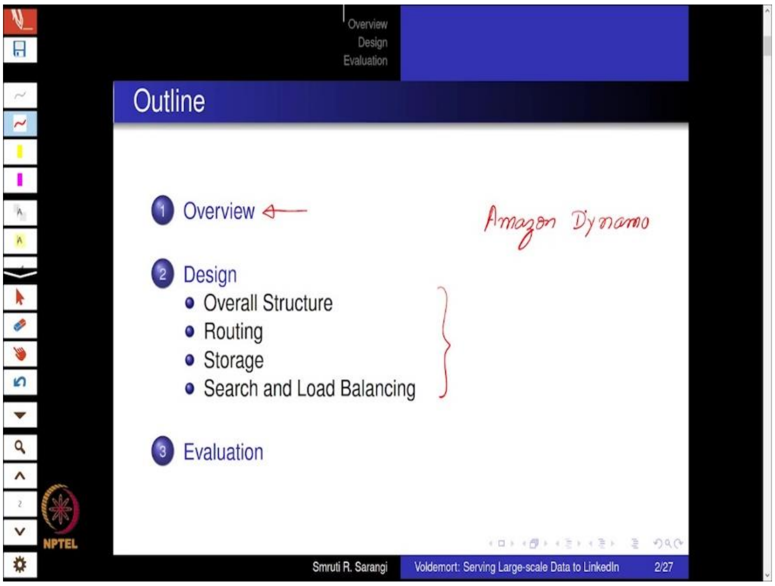
Smruti R. Sarangi

Department of Computer Science
Indian Institute of Technology
New Delhi, India

Smruti R. Sarangi | Voldemort: Serving Large-scale Data to LinkedIn | 1/27

In this lecture, we will discuss the Voldemort project. So, the Voldemort Project is an open-source project. And, but it has some important modifications were made to the open-source project to serve large scale data to LinkedIn. So, LinkedIn, as we all know, is a professional networking site. And so, LinkedIn engineers, were working on many modifications, they worked on Voldemort, and Voldemort later on was open source. But the important aspect of this lecture is not to discuss Voldemort. Rather, it is to discuss modifications made to Voldemort, for LinkedIn's particular specific needs.

(Refer Slide Time: 1:06)



Overview
Design
Evaluation

Outline

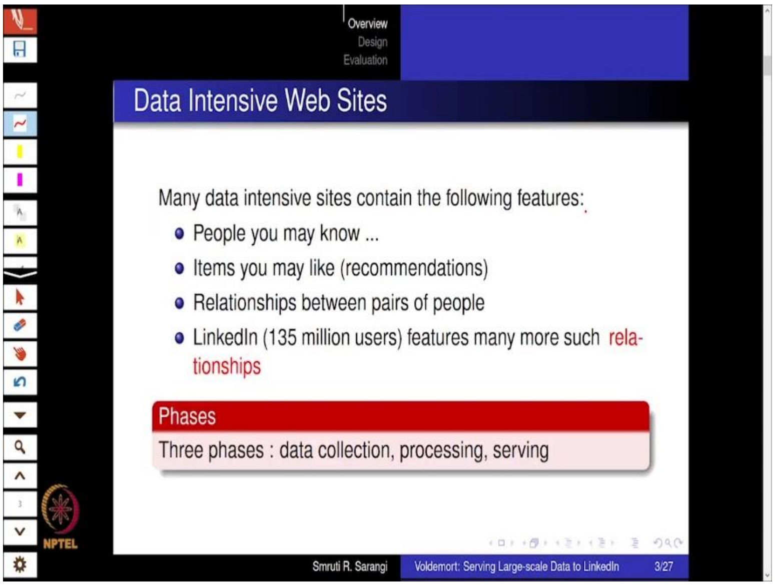
1. Overview
2. Design
   - Overall Structure
   - Routing
   - Storage
   - Search and Load Balancing
3. Evaluation

Amazon Dynamo

Smruti R. Sarangi | Voldemort: Serving Large-scale Data to LinkedIn | 2/27

So, we will first discuss the overview of the design, the overall structure and evaluation. So, of course, in this case, we will rely heavily on the Amazon Dynamo paper. So, this kind of builds on Amazon Dynamo. So, it is important that viewers take a look at the lecture on Amazon dynamo, which is there in this lecture series is there in this playlist. But that is a very important prerequisite for this. Because without understanding the dynamics of Dynamo, they will not be able to follow this paper.

(Refer Slide Time: 1:51)



Overview
Design
Evaluation

Data Intensive Web Sites

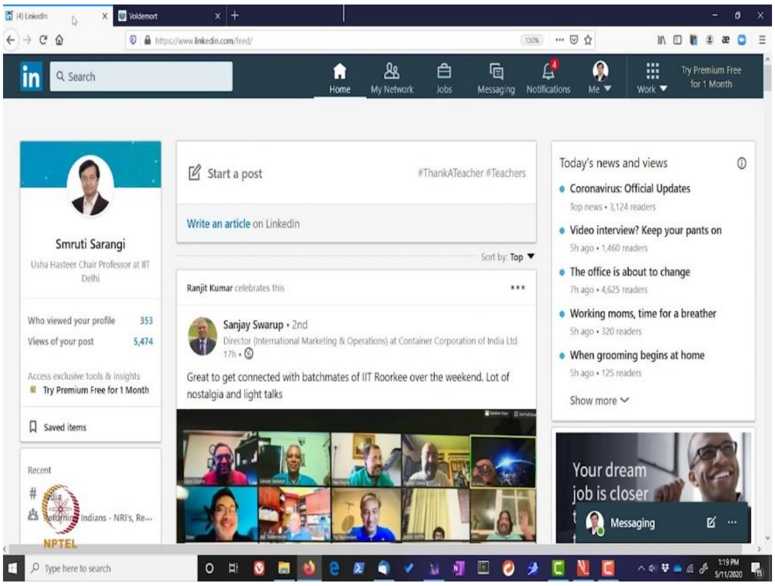
Many data intensive sites contain the following features:

- People you may know ...
- Items you may like (recommendations)
- Relationships between pairs of people
- LinkedIn (135 million users) features many more such relationships

Phases

Three phases : data collection, processing, serving

Smruti R. Sarangi | Voldemort: Serving Large-scale Data to LinkedIn | 3/27



Smruti Sarangi
Usha Hasteer Chair Professor at IIT Delhi

Who viewed your profile	353
Views of your post	5,474

Access exclusive tools & insights
Try Premium Free for 1 Month
Saved items

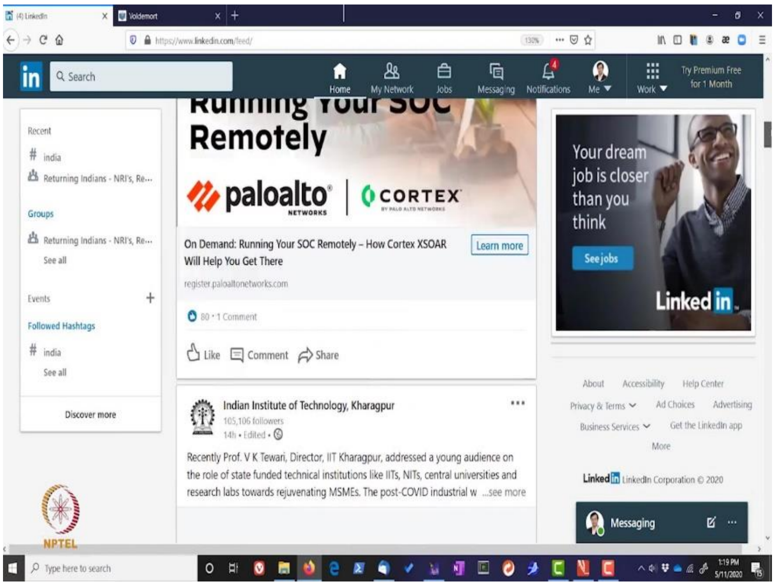
Recent
india
Returning Indians - NRI's, Re...

Start a post #ThankATeacher #Teachers
Write an article on LinkedIn
Sort by: Top

Ranjit Kumar celebrates this
Sanjay Swarup • 2nd
Director (International Marketing & Operations) at Container Corporation of India Ltd
17h •
Great to get connected with batchmates of IIT Roorkee over the weekend. Lot of nostalgia and light talks

Today's news and views
- Coronavirus: Official Updates
  Top news • 3,124 readers
- Video interview? Keep your pants on
  5h ago • 1,460 readers
- The office is about to change
  7h ago • 4,625 readers
- Working moms, time for a breather
  5h ago • 320 readers
- When grooming begins at home
  5h ago • 125 readers

Show more

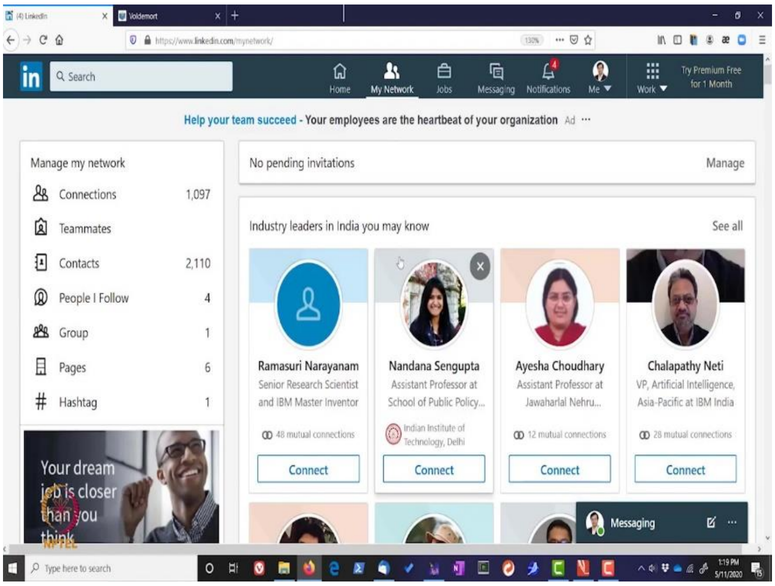


Recent
india
Returning Indians - NRI's, Re...
Groups
Returning Indians - NRI's, Re...
See all
Events
Followed Hashtags
india
See all
Discover more

On Demand: Running Your SOC Remotely – How Cortex XSOAR Will Help You Get There
Learn more
register.paloaltonetworks.com
80 • 1 Comment
Like Comment Share

Indian Institute of Technology, Kharagpur
105,106 followers
14h • Edited •
Recently Prof. V K Tewari, Director, IIT Kharagpur, addressed a young audience on the role of state funded technical institutions like IITs, NITs, central universities and research labs towards rejuvenating MSMEs. The post-COVID industrial w ...see more

About Accessibility Help Center
Privacy & Terms Ad Choices Advertising
Business Services Get the LinkedIn app
More
LinkedIn LinkedIn Corporation © 2020



Help your team succeed - Your employees are the heartbeat of your organization Ad

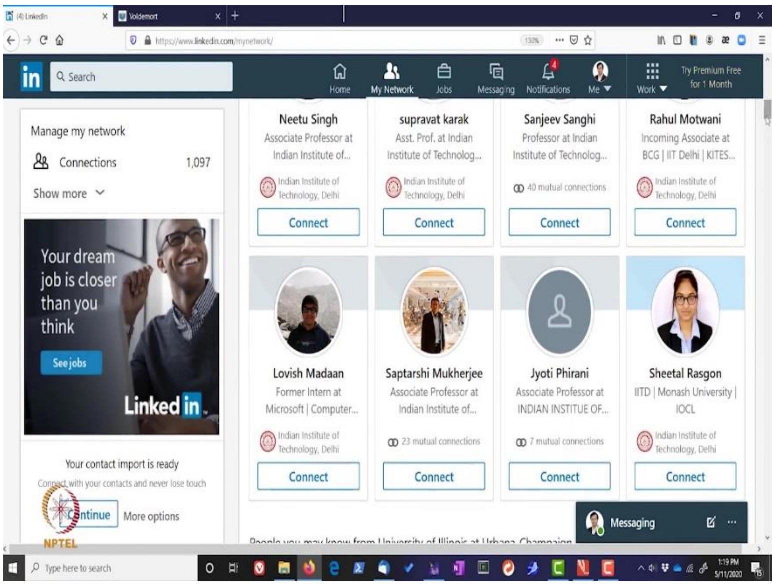
Manage my network

Connections	1,097
Teammates	
Contacts	2,110
People I Follow	4
Group	1
Pages	6
Hashtag	1

No pending invitations Manage

Industry leaders in India you may know See all

Ramasuri Narayanam	Nandana Sengupta	Ayesha Choudhary	Chalapathy Neti
Senior Research Scientist and IBM Master Inventor	Assistant Professor at School of Public Policy...	Assistant Professor at Jawaharlal Nehru...	VP, Artificial Intelligence, Asia-Pacific at IBM India
48 mutual connections	Indian Institute of Technology, Delhi	12 mutual connections	28 mutual connections
Connect	Connect	Connect	Connect

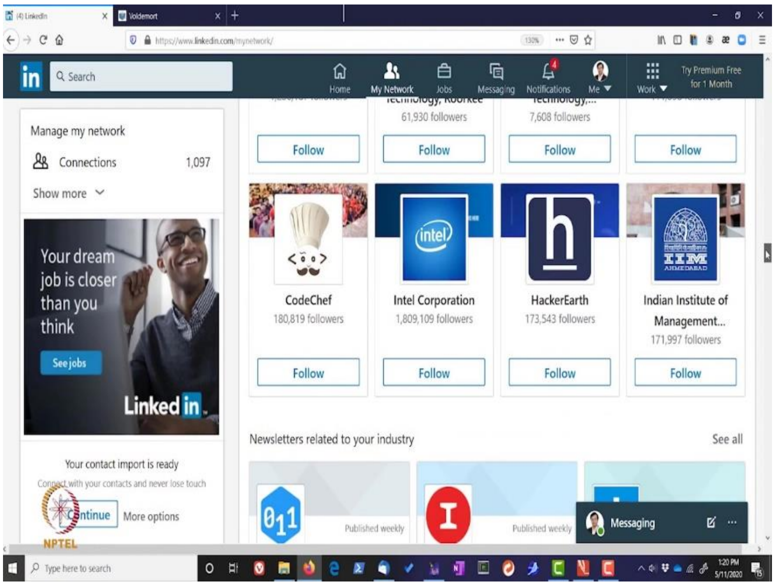

(4) LinkedIn
Voldemort
https://www.linkedin.com/mynetwork/
Search
Home
My Network
Jobs
Messaging
Notifications
Me
Work
Try Premium Free for 1 Month
Manage my network
Connections 1,097
Show more
Your dream job is closer than you think
See jobs
Linked in
Your contact import is ready
Connect with your contacts and never lose touch
Continue
More options
Neetu Singh
Associate Professor at Indian Institute of...
Indian Institute of Technology, Delhi
Connect
supravat karak
Asst. Prof. at Indian Institute of Technolog...
Indian Institute of Technology, Delhi
Connect
Sanjeev Sanghi
Professor at Indian Institute of Technolog...
40 mutual connections
Connect
Rahul Motwani
Incoming Associate at BCG | IIT Delhi | KITES...
Indian Institute of Technology, Delhi
Connect
Lovish Madaan
Former Intern at Microsoft | Computer...
Indian Institute of Technology, Delhi
Connect
Saptarshi Mukherjee
Associate Professor at Indian Institute of...
23 mutual connections
Connect
Jyoti Phirani
Associate Professor at INDIAN INSTITUE OF...
7 mutual connections
Connect
Sheetal Rasgon
IITD | Monash University | IOCL
Indian Institute of Technology, Delhi
Connect
Messaging
People you may know from University of Illinois at Urbana-Champaign
NPTEL
Type here to search
1:19 PM
5/11/2020

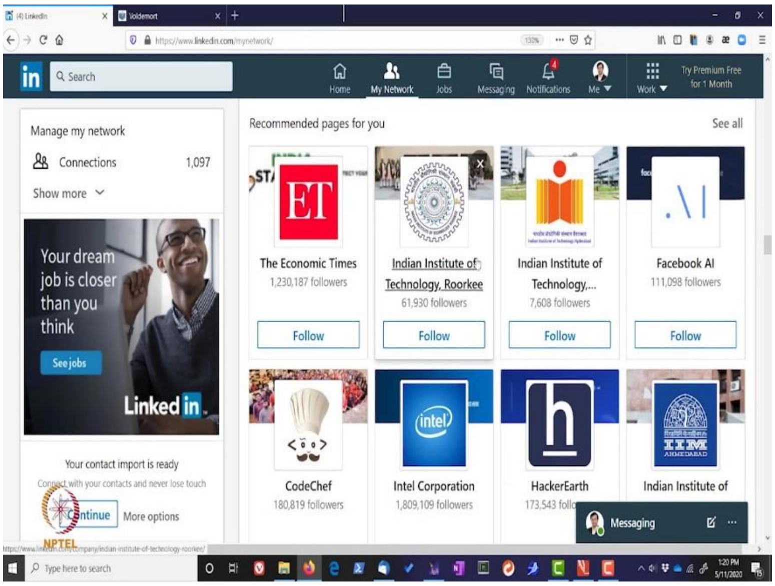
So, many data intensive sites contain the following features. So, I will discuss a few features of LinkedIn by actually showing you my LinkedIn site. So, this site that you see over here is my LinkedIn site. And that is me. So, LinkedIn, of course, does a lot of things like we have some news over here regarding Coronavirus, and so on and all the posts on my site.

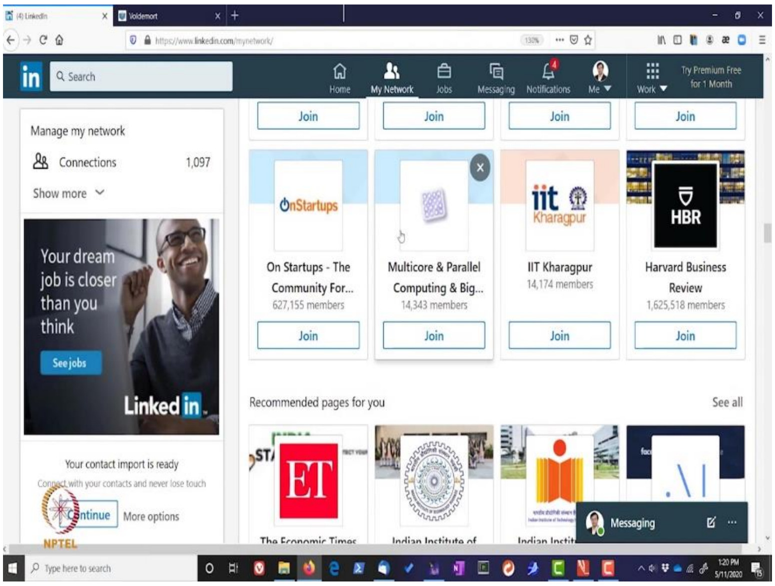
So, I would like to take you to my network. And this is the way that my network looks now. So, the most important thing, one of the key things that LinkedIn actually does, is that it suggests a name of people, a lot of people that I may know, I might know. And this is actually the way that I make connections.

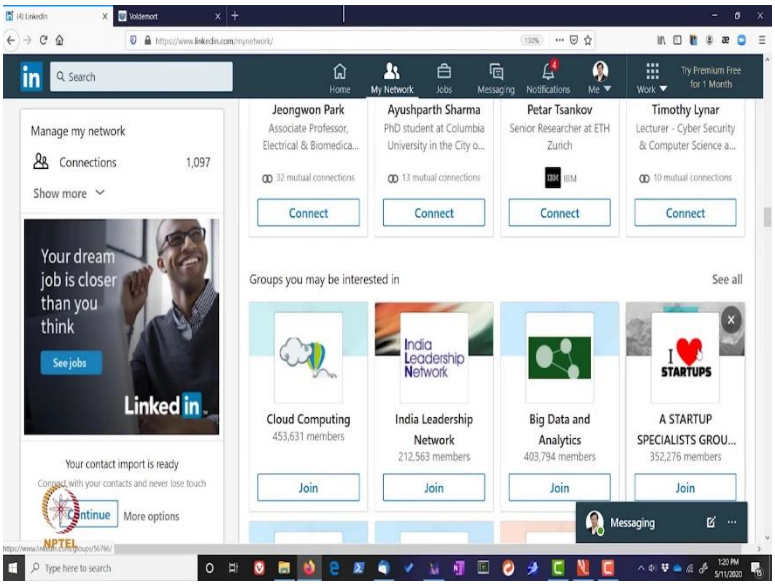
So, let us say that suggests a lot of people that I may possibly know. And in that so I can essentially look through them and then I can make connections. So how does it know them? Well, it is kind of mind my behavior. So, from my behavior, it has some idea of the kind of people that I like to connect with. For example, I work on cybersecurity.

(Refer Slide Time: 3:05)



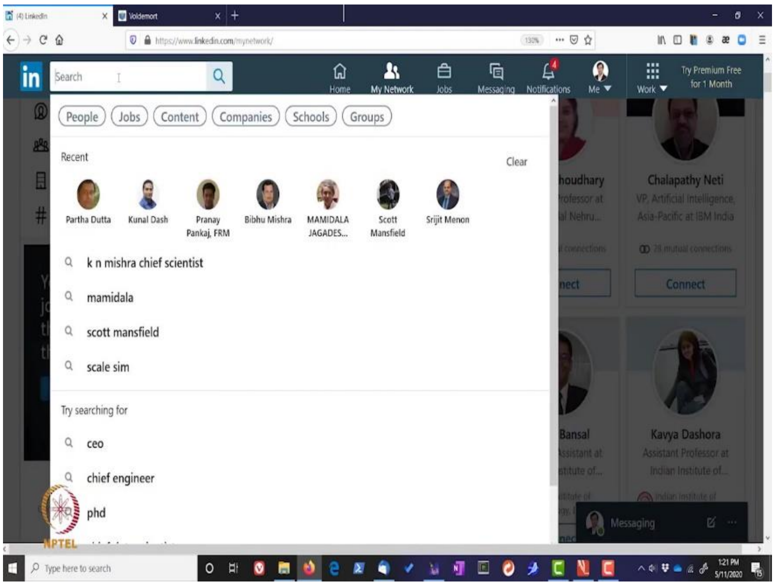


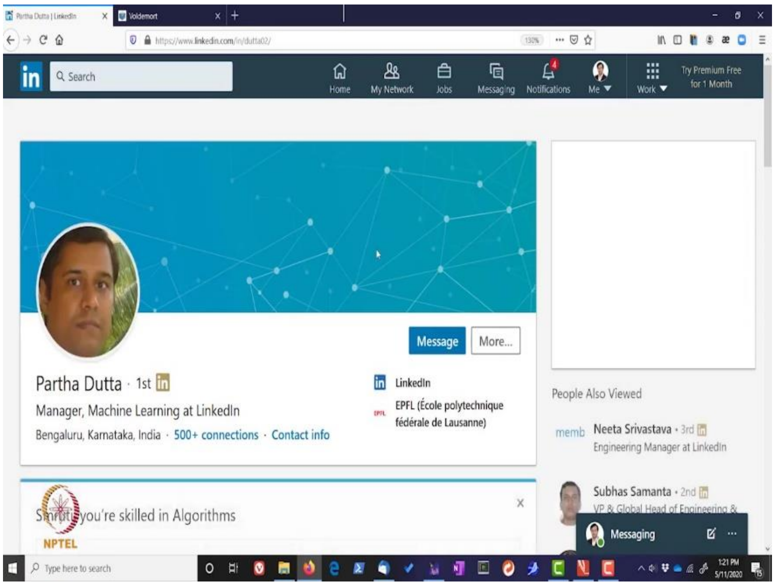




So, it shows me cybersecurity people. And also, I work in computer architecture. So, it is showing me Intel’s site, I read sources, just look at how much LinkedIn tracks my activity. So, I read economic news quite a bit. So, the economic times shows up over here. I work on multicore computing, multicore shows up over here, I am from IIT Kharagpur, IIT Kharagpur shows up over here. And I also mentored some startups. So, I love startups also shows up over here. So it is like my entire personality is reflected over here. So, find all of this data, LinkedIn had to do a lot of data mining. So, this is the people you may know.

(Refer Slide Time: 3:48)





Partha Dutta · 1st
Manager, Machine Learning at LinkedIn
Bengaluru, Karnataka, India · 500+ connections · Contact info

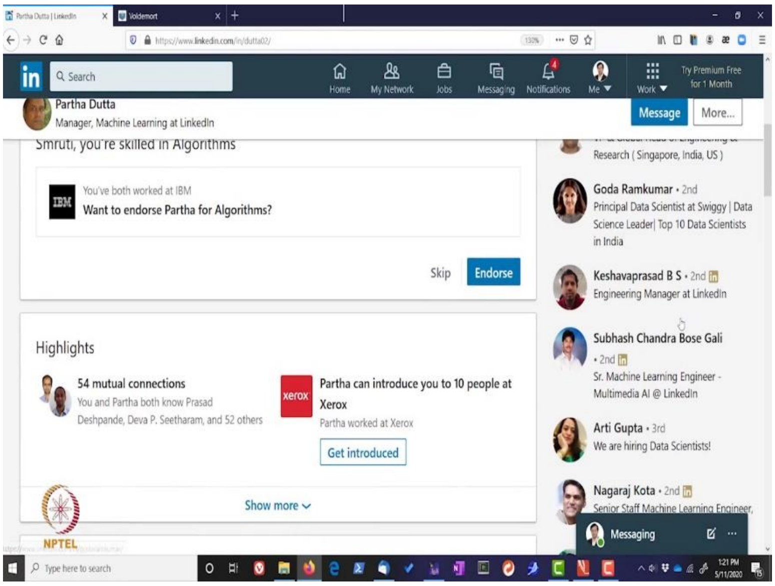
LinkedIn
EPFL (École polytechnique fédérale de Lausanne)

Smruti, you're skilled in Algorithms

People Also Viewed

Neeta Srivastava • 3rd
Engineering Manager at LinkedIn

Subhas Samanta • 2nd
VP & Global Head of Engineering &



Partha Dutta
Manager, Machine Learning at LinkedIn

Smruti, you're skilled in Algorithms

You've both worked at IBM
Want to endorse Partha for Algorithms?

Highlights

54 mutual connections
You and Partha both know Prasad Deshpande, Deva P. Seetharam, and 52 others

Partha can introduce you to 10 people at Xerox
Partha worked at Xerox

Research (Singapore, India, US)

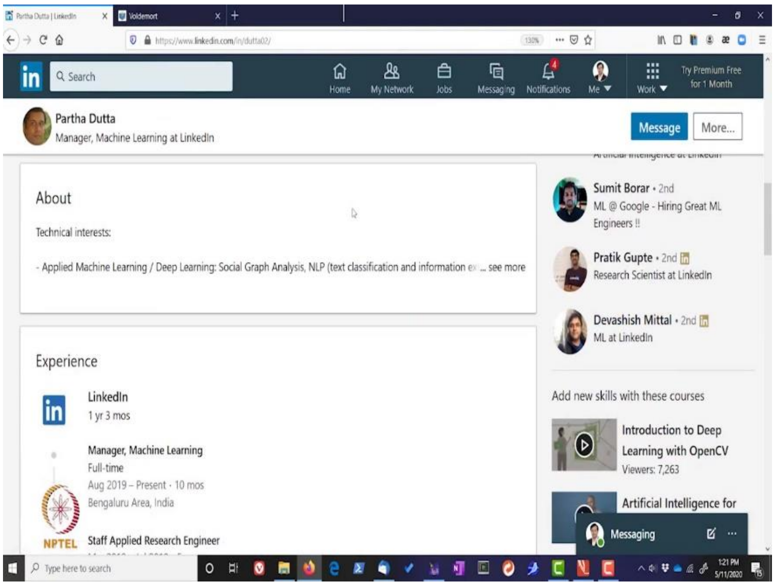
Goda Ramkumar • 2nd
Principal Data Scientist at Swiggy | Data Science Leader| Top 10 Data Scientists in India

Keshavaprasad B S • 2nd
Engineering Manager at LinkedIn

Subhash Chandra Bose Gali • 2nd
Sr. Machine Learning Engineer - Multimedia AI @ LinkedIn

Arti Gupta • 3rd
We are hiring Data Scientists!

Nagaraj Kota • 2nd
Senior Staff Machine Learning Engineer,



Partha Dutta
Manager, Machine Learning at LinkedIn

About

Technical interests:

- Applied Machine Learning / Deep Learning: Social Graph Analysis, NLP (text classification and information ex ... see more

Experience

LinkedIn
1 yr 3 mos

Manager, Machine Learning
Full-time
Aug 2019 – Present · 10 mos
Bengaluru Area, India

Staff Applied Research Engineer

Artificial Intelligence at LinkedIn

Sumit Borar • 2nd
ML @ Google - Hiring Great ML Engineers !!

Pratik Gupte • 2nd
Research Scientist at LinkedIn

Devashish Mittal • 2nd
ML at LinkedIn

Add new skills with these courses

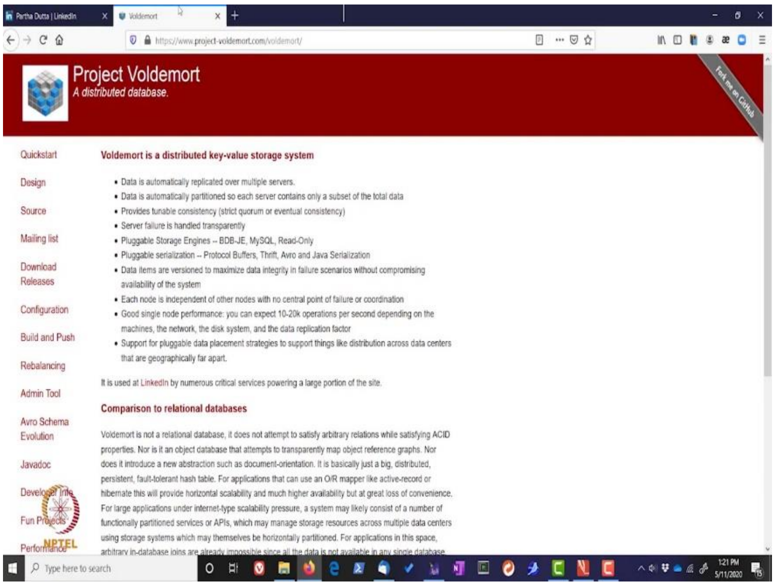
Introduction to Deep Learning with OpenCV
Viewers: 7,263

Artificial Intelligence for

Another thing that LinkedIn actually does is that it does something called collaborative filtering. So, let me go to the website of my friend, Partha Dutta, who is also a very acclaimed researcher in distributed systems. So, incidentally, he also works at LinkedIn. And the important thing is to look at something on the right, that people also viewed.

So, this is also telling me that people who have seen Partha Dutta site have also viewed other similar people. And so, these are all of his connections, so I might want to connect with them. So, this is a collaborative filtering. You can think of it this way, that it is also telling me that, others have also viewed Partha Dutta, and I might want to view them.

(Refer Slide Time: 4:36)



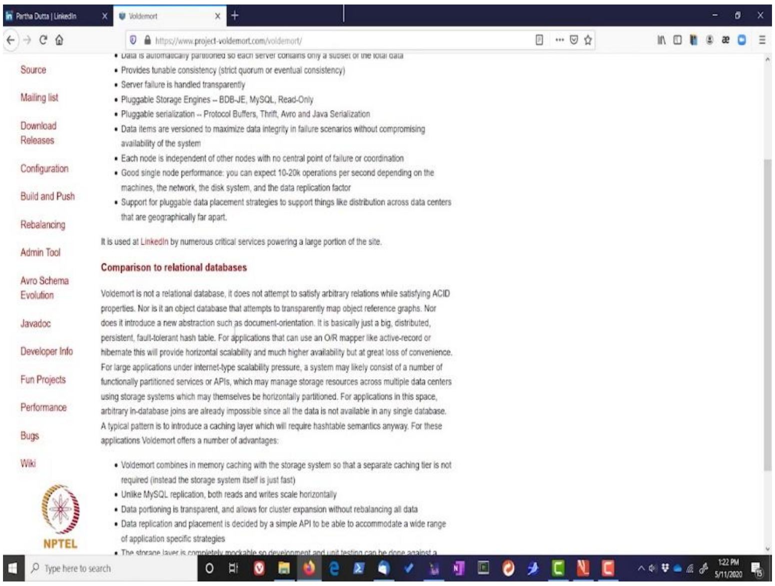
Voldemort is a distributed key-value storage system

- Data is automatically replicated over multiple servers.
- Data is automatically partitioned so each server contains only a subset of the total data
- Provides tunable consistency (strict quorum or eventual consistency)
- Server failure is handled transparently
- Pluggable Storage Engines -- BDB-JE, MySQL, Read-Only
- Pluggable serialization -- Protocol Buffers, Thrift, Avro and Java Serialization
- Data items are versioned to maximize data integrity in failure scenarios without compromising availability of the system
- Each node is independent of other nodes with no central point of failure or coordination
- Good single node performance: you can expect 10-20k operations per second depending on the machines, the network, the disk system, and the data replication factor
- Support for pluggable data placement strategies to support things like distribution across data centers that are geographically far apart.

It is used at LinkedIn by numerous critical services powering a large portion of the site.

Comparison to relational databases

Voldemort is not a relational database, it does not attempt to satisfy arbitrary relations while satisfying ACID properties. Nor is it an object database that attempts to transparently map object reference graphs. Nor does it introduce a new abstraction such as document-orientation. It is basically just a big, distributed, persistent, fault-tolerant hash table. For applications that can use an O/R mapper like active-record or hibernate this will provide horizontal scalability and much higher availability but at great loss of convenience. For large applications under internet-type scalability pressure, a system may likely consist of a number of functionally partitioned services or APIs, which may manage storage resources across multiple data centers using storage systems which may themselves be horizontally partitioned. For applications in this space, arbitrary in-database joins are already impossible since all the data is not available in any single database.



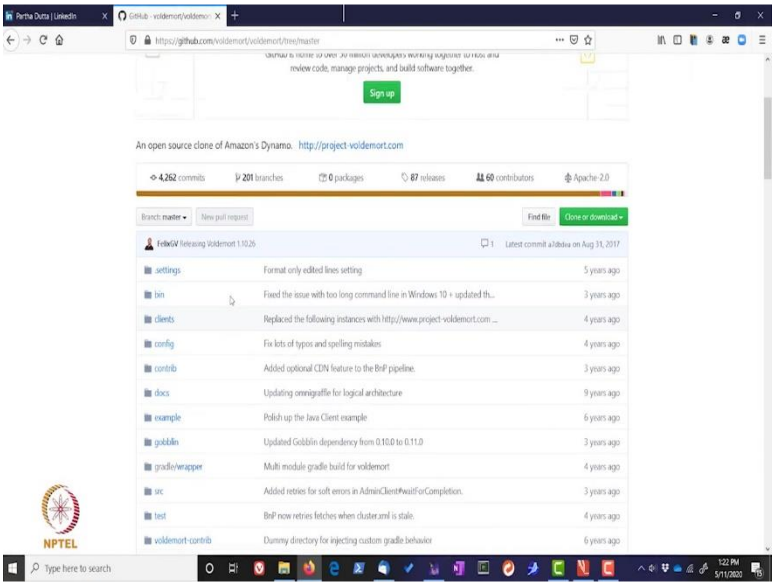
- Data is automatically partitioned so each server contains only a subset of the total data
- Provides tunable consistency (strict quorum or eventual consistency)
- Server failure is handled transparently
- Pluggable Storage Engines -- BDB-JE, MySQL, Read-Only
- Pluggable serialization -- Protocol Buffers, Thrift, Avro and Java Serialization
- Data items are versioned to maximize data integrity in failure scenarios without compromising availability of the system
- Each node is independent of other nodes with no central point of failure or coordination
- Good single node performance: you can expect 10-20k operations per second depending on the machines, the network, the disk system, and the data replication factor
- Support for pluggable data placement strategies to support things like distribution across data centers that are geographically far apart.

It is used at LinkedIn by numerous critical services powering a large portion of the site.

Comparison to relational databases

Voldemort is not a relational database, it does not attempt to satisfy arbitrary relations while satisfying ACID properties. Nor is it an object database that attempts to transparently map object reference graphs. Nor does it introduce a new abstraction such as document-orientation. It is basically just a big, distributed, persistent, fault-tolerant hash table. For applications that can use an O/R mapper like active-record or hibernate this will provide horizontal scalability and much higher availability but at great loss of convenience. For large applications under internet-type scalability pressure, a system may likely consist of a number of functionally partitioned services or APIs, which may manage storage resources across multiple data centers using storage systems which may themselves be horizontally partitioned. For applications in this space, arbitrary in-database joins are already impossible since all the data is not available in any single database. A typical pattern is to introduce a caching layer which will require hashtable semantics anyway. For these applications Voldemort offers a number of advantages:

- Voldemort combines in memory caching with the storage system so that a separate caching tier is not required (instead the storage system itself is just fast)
- Unlike MySQL replication, both reads and writes scale horizontally
- Data portioning is transparent, and allows for cluster expansion without rebalancing all data
- Data replication and placement is decided by a simple API to be able to accommodate a wide range of application specific strategies
- The storage layer is completely mockable so development and unit testing can be done against a



https://github.com/voldemort/voldemort/tree/master

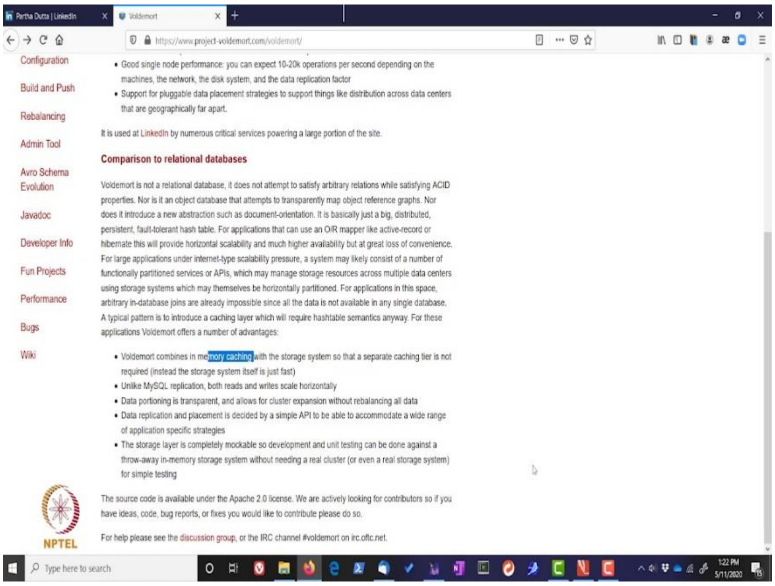
GitHub is home to over 50 million developers working together to host and review code, manage projects, and build software together.

An open source clone of Amazon's Dynamo. http://project-voldemort.com

4,262 commits · 201 branches · 0 packages · 87 releases · 60 contributors · Apache-2.0

FelixGV Releasing Voldemort 1.10.26 · 1 · Latest commit a7dbdea on Aug 31, 2017

.settings	Format only edited lines setting	5 years ago
bin	Fixed the issue with too long command line in Windows 10 + updated th...	3 years ago
clients	Replaced the following instances with http://www.project-voldemort.com ...	4 years ago
config	Fix lots of typos and spelling mistakes	4 years ago
contrib	Added optional CDN feature to the BnP pipeline.	3 years ago
docs	Updating omnigraffle for logical architecture	9 years ago
example	Polish up the Java Client example	6 years ago
gobblin	Updated Gobblin dependency from 0.10.0 to 0.11.0	3 years ago
gradle/wrapper	Multi module gradle build for voldemort	4 years ago
src	Added retries for soft errors in AdminClient#waitForCompletion.	3 years ago
test	BnP now retries fetches when cluster.xml is stale.	4 years ago
voldemort-contrib	Dummy directory for injecting custom gradle behavior	6 years ago



https://www.project-voldemort.com/voldemort/

- Good single node performance: you can expect 10-20k operations per second depending on the machines, the network, the disk system, and the data replication factor
- Support for pluggable data placement strategies to support things like distribution across data centers that are geographically far apart.

It is used at LinkedIn by numerous critical services powering a large portion of the site.

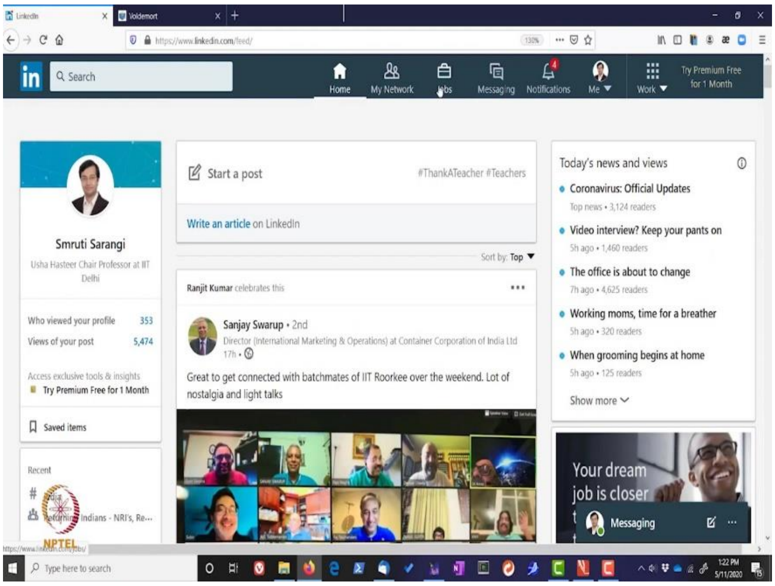
Comparison to relational databases

Voldemort is not a relational database, it does not attempt to satisfy arbitrary relations while satisfying ACID properties. Nor is it an object database that attempts to transparently map object reference graphs. Nor does it introduce a new abstraction such as document-orientation. It is basically just a big, distributed, persistent, fault-tolerant hash table. For applications that can use an O/R mapper like active-record or hibernate this will provide horizontal scalability and much higher availability but at great loss of convenience. For large applications under internet-type scalability pressure, a system may likely consist of a number of functionally partitioned services or APIs, which may manage storage resources across multiple data centers using storage systems which may themselves be horizontally partitioned. For applications in this space, arbitrary in-database joins are already impossible since all the data is not available in any single database. A typical pattern is to introduce a caching layer which will require hashtable semantics anyway. For these applications Voldemort offers a number of advantages:

- Voldemort combines in memory caching with the storage system so that a separate caching tier is not required (instead the storage system itself is just fast)
- Unlike MySQL replication, both reads and writes scale horizontally
- Data portioning is transparent, and allows for cluster expansion without rebalancing all data
- Data replication and placement is decided by a simple API to be able to accommodate a wide range of application specific strategies
- The storage layer is completely mockable so development and unit testing can be done against a throw-away in-memory storage system without needing a real cluster (or even a real storage system) for simple testing

The source code is available under the Apache 2.0 license. We are actively looking for contributors so if you have ideas, code, bug reports, or fixes you would like to contribute please do so.

For help please see the discussion group, or the IRC channel #voldemort on irc.oftc.net.



https://www.linkedin.com/feed/

Smruti Sarangi

Usha Hasteer Chair Professor at IIT Delhi

Who viewed your profile	353
Views of your post	5,474

Access exclusive tools & insights

Try Premium Free for 1 Month

Saved items

Recent

Returning Indians - NRI's, Re...

Start a post · #ThankATeacher #Teachers

Write an article on LinkedIn

Sort by: Top

Ranjit Kumar celebrates this

Sanjay Swarup • 2nd

Director (International Marketing & Operations) at Container Corporation of India Ltd

17h •

Great to get connected with batchmates of IIT Roorkee over the weekend. Lot of nostalgia and light talks

Today's news and views

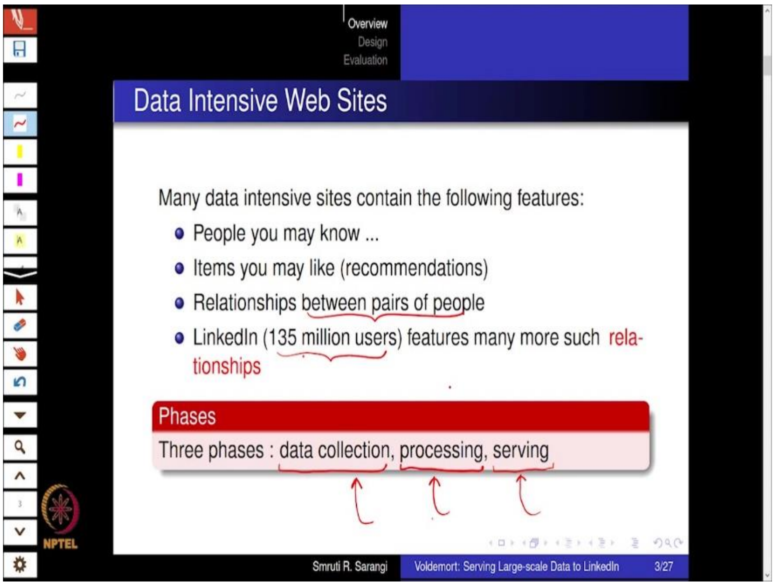
- Coronavirus: Official Updates
  Top news • 3,124 readers
- Video interview? Keep your pants on
  5h ago • 1,460 readers
- The office is about to change
  7h ago • 4,625 readers
- Working moms, time for a breather
  5h ago • 320 readers
- When grooming begins at home
  5h ago • 125 readers

Show more

And the key behind all of this is, of course, the Voldemort project, which is a distributed database. It is a key value store. Similar to the other key value stores that we have actually seen. The it is open source. So, the source code of this project is available on GitHub. And this source code can be used. We can make several configuration changes Voldemort has in memory caching, replication, partitioning all of that.

So, what we will see is how was this made LinkedIn friendly? For the two things that I discussed? Which was that, how is my network managed? In a sense? How does LinkedIn suggest that I connect with new people? And also, how are my contacts managed? Let us say if I were to go to a contact, how does it show me that who were all the other people who have been viewing my contact. So, given that I have seen this, let me go back to the slide that I was showing over here.

(Refer Slide Time: 5:43)



Data Intensive Web Sites

Many data intensive sites contain the following features:

- People you may know ...
- Items you may like (recommendations)
- Relationships between pairs of people
- LinkedIn (135 million users) features many more such relationships

Phases

Three phases : data collection, processing, serving

And let us restart the presentation. So, now that you have seen that LinkedIn makes a lot of recommendations. So, Amazon also does that. So, Amazon suggests items that I may buy, LinkedIn suggests people that I may connect with, it looks at relationships between pairs of people.

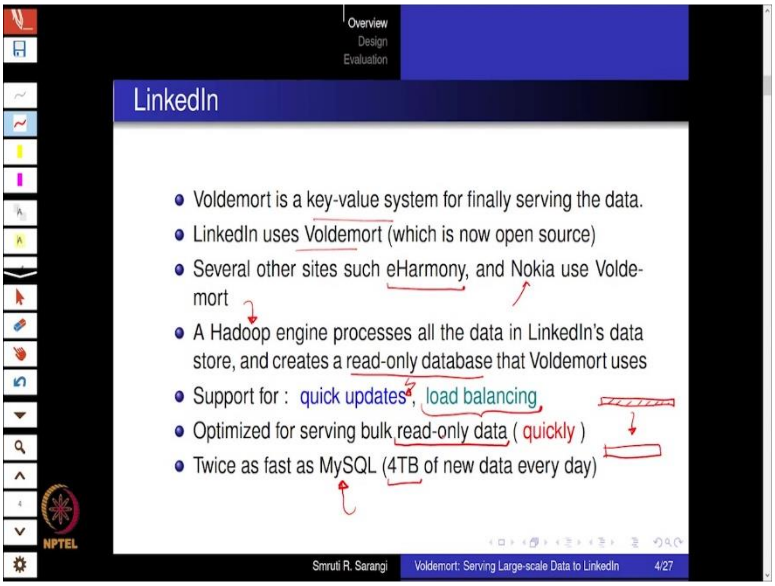
So, as of the date of publication of this paper, which was quite a while ago, we had 135 million LinkedIn users. Now, of course, we will have much more. And also, we will have many, many relationships among these people. And these relationships have to be mined. So, we have to collect the data, we have to process the data. So, all of this data about mean is to be collected. So, as you just saw, everything that I do, via LinkedIn is kind of tracked.

And by that LinkedIn has an idea of what is it that I like, what is it that that dislike, so the fact that I like economic news, maybe my wife does not know, but LinkedIn knows. So, this is to the extent that online sites have an idea about us, which is okay, in the modern world. And that is how they make recommendations to me, which means there is a huge amount of data collection, all of this data is processed.

And when I open the site, all of this data is served to me. And I can quickly see, so of course, all three of these tasks are not happening at the same time. Instead, they are happening at different points in time. And that is the wonderful part of this. And that is why LinkedIn is such a successful site, such a successful professional networking site. And as you have just seen, I use it a lot. And I love using it.

And the reason is that a lot of information is given to me regarding what I should read, who I should connect with, what are their connections, what is the relationship between the connections, I kind of get to see all of that.

(Refer Slide Time: 8:04)



Overview
Design
Evaluation

LinkedIn

- Voldemort is a key-value system for finally serving the data.
- LinkedIn uses Voldemort (which is now open source)
- Several other sites such eHarmony, and Nokia use Voldemort
- A Hadoop engine processes all the data in LinkedIn's data store, and creates a read-only database that Voldemort uses
- Support for : quick updates, load balancing
- Optimized for serving bulk read-only data (quickly)
- Twice as fast as MySQL (4TB of new data every day)

Smruti R. Sarangi   Voldemort: Serving Large-scale Data to LinkedIn   4/27

So, Voldemort is a key value system. It is now open source, LinkedIn uses it. But Voldemort is used by several other sites. It is used by eHarmony, which is a very common matchmaking site in the US. Even Nokia also used to use Voldemort, I am not sure about it right now. So, Voldemort has an internal Hadoop engine, which is a MapReduce engine, which processes all the data in LinkedIn data store.

And it essentially creates a read only database that Voldemort uses. So, what is the key idea? Well, the key idea is that outside the Voldemort system, we have a Hadoop system that takes in all the information, processes it, makes it ready, and creates a kind of a read only version of data. Gives it to Voldemort such that Voldemort can serve it to customers.

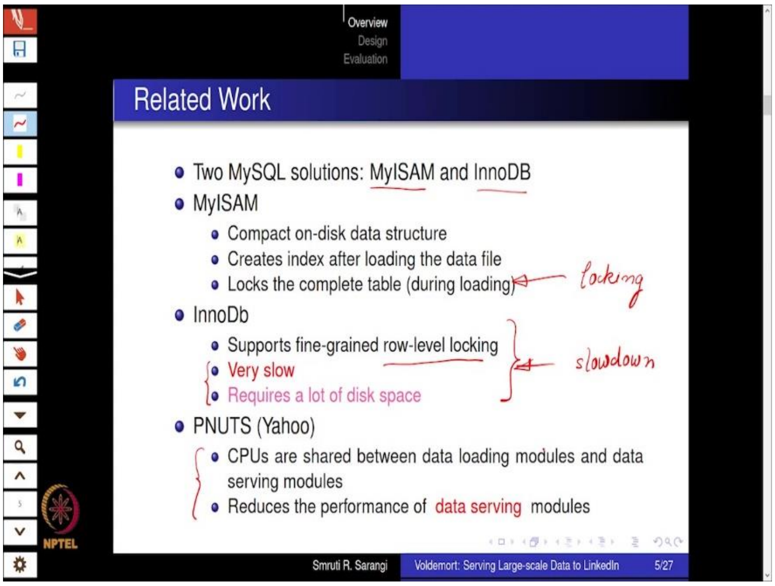
So, what we actually need is we need a quick update system, where Voldemort can quickly read whatever the Hadoop system is generating for it. And we need load balancing, we do not want any server to get overloaded with data, which has been a constant concern throughout the designs of this course.

So, the main aim over here, so if you would see our last lecture was on Facebook's photo storage. So, the photo storage was mostly read only data. But this data is even more read only in the sense that it is not modified in the sense that I cannot go and let us say modify the list of recommendations that LinkedIn is making. In fact, nobody can, only LinkedIn can. So, all of this data is totally read only.

So, what happens is that the Hadoop system produces a tranche of data and the entire data is sort of bulk loaded into Voldemort. And this and all of this data is read only. And so, the all of this data is read only, so it comes into Voldemort. And we need to ensure that this is much faster than existing solutions.

An existing solution, the default solution is a MySQL database. And LinkedIn system is reasonably faster twice as fast as MySQL. So, we load 4 terabytes of new data every day and so, that was the date of publication of the paper. And nowadays, it must be much more.

(Refer Slide Time: 10:45)



Overview
Design
Evaluation

Related Work

- Two MySQL solutions: MyISAM and InnoDB
- MyISAM
  - Compact on-disk data structure
  - Creates index after loading the data file
  - Locks the complete table (during loading) ← locking
- InnoDb
  - Supports fine-grained row-level locking
  - Very slow
  - Requires a lot of disk space

  } ← slowdown
- PNUTS (Yahoo)
  - CPUs are shared between data loading modules and data serving modules
  - Reduces the performance of data serving modules

Smruti R. Sarangi   Voldemort: Serving Large-scale Data to LinkedIn   5/27

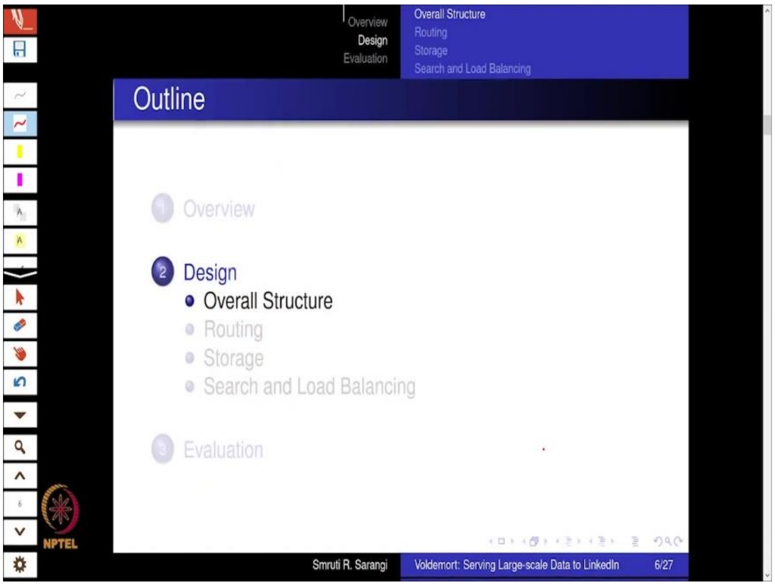
So, there are two MySQL solutions. So, MySQL has two native data storage formats, MyISAM and InnoDB. So, MyISAM is a compact on disk data structure. So, it creates an index after loading the data file. So, the even though the MyISAM system is reasonably fast, and it can be used for this purpose, the main problem is that it locks the complete table during the process of loading.

So, this process of locking the entire data set unnecessarily causes a disruption in the quality of service and this is something that is clearly not desirable. So, this is not something that you would like the other option is InnoDB. So, InnoDB supports far more fine-grained row level locking.

So, it is possible to maybe read some data and whatever we are reading and updating only those rows can be locked. The main problem is a rather slow process, it also requires a lot of disk space. The main reason being that since rows are being accessed kind of individually, we need a lot of disk space to actually store all the indexing data structures.

So, here also there is a overhead of locking. And in InnoDB, there is an overhead of, excessive slowdown because of indexing. So, then Yahoo has a PNUTS system, where CPUs are shared between the data loading modules and the data serving modules. So, even though this sounds is a good idea, in the sense that we can have a very flexible structure, where there are some data loaders, and there are some data servers, it reduces the performance as a whole. So, the LinkedIn engineers did evaluate all of these solutions. And then they came up with a system of their own.

(Refer Slide Time: 12:56)



Overview
Design
Evaluation

Overall Structure
Routing
Storage
Search and Load Balancing

Outline

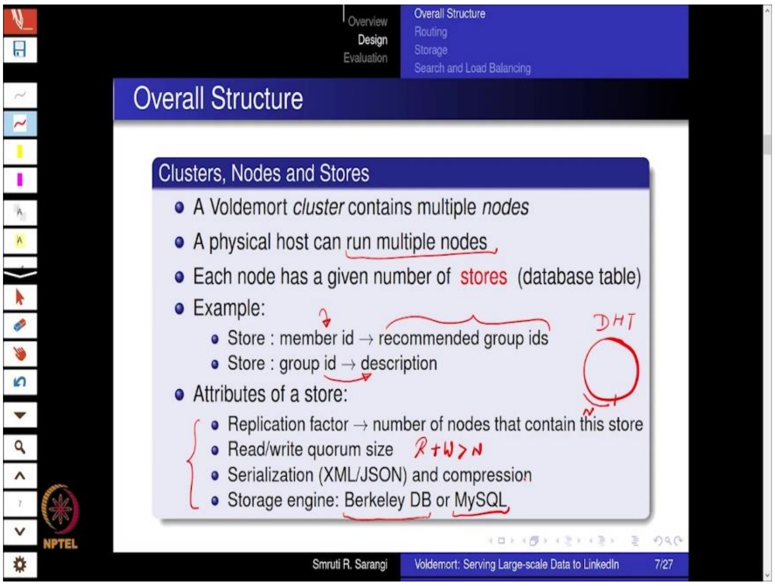
1 Overview

2 Design
- Overall Structure
- Routing
- Storage
- Search and Load Balancing

3 Evaluation

Smruti R. Sarangi

Voldemort: Serving Large-scale Data to LinkedIn 6/27



Overall Structure

Clusters, Nodes and Stores

- A Voldemort *cluster* contains multiple *nodes*
- A physical host can run multiple nodes
- Each node has a given number of stores (database table)
- Example:
  - Store : member id → recommended group ids
  - Store : group id → description
- Attributes of a store:
  - Replication factor → number of nodes that contain this store
  - Read/write quorum size $R+W>N$
  - Serialization (XML/JSON) and compression
  - Storage engine: Berkeley DB or MySQL

DHT

N

Smruti R. Sarangi

Voldemort: Serving Large-scale Data to LinkedIn

7/27

So, the overall structure is like this, that Voldemort cluster contains multiple nodes. And a physical host in the multiple nodes. So, these are like virtual nodes in chord or virtual nodes in Dynamo. And a physical host can run multiple nodes. Each node has a given number of stores or database tables. And each stores some information.

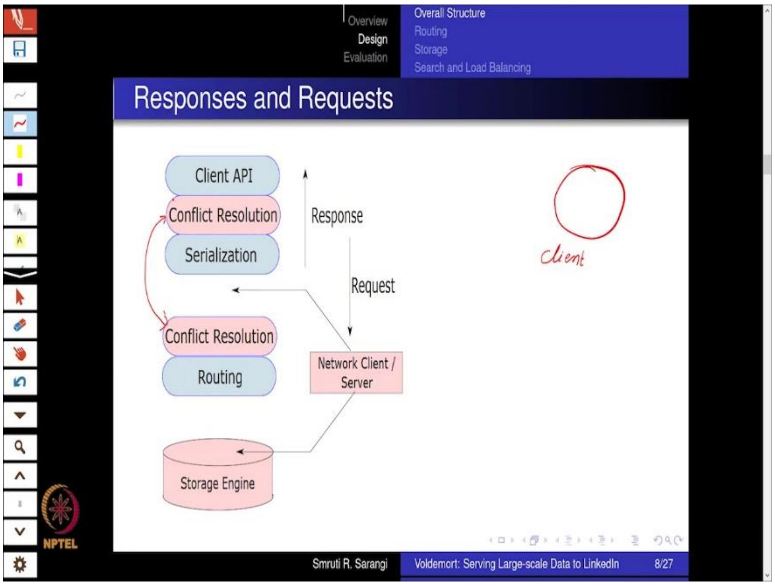
So, what is the information? Well, if I saw, let us say that for a given member ID, which is me if it is recommending a couple of LinkedIn groups, for example, then it can store the list of recommended LinkedIn groups, the recommended Group IDs, and also for each group ID it can store a description. And so, then the attributes of a store can be the replication factor on how many physical machines is a store actually replicated? The size of a read write Quorum?

So, I am deliberately going fast on this, because read quorum, write quorum, we have all discussed in great detail when we were discussing the other projects like percolator and dynamo, where the read write quorum, where the basic idea was discussed. And recall that if R + W > N, where R is the size of the read quorum, W is the size of the write quorum, and N is the number of replicated notes.

So, what do we do? Well, the baseline is that this uses a DHT similar to dynamo so on almost all of these cases, a DHT is used. Of course, we have replication in the sense that it is a chord like system, but it is a one hop DHT, and every node is assigned to N of its successors subject to the successors being on different machines.

And within the successors, we call it a preference list. Again, harkens back to Dynamo. So, I would request the readers to take a look at Dynamo. And of course, R + W > N. So, for transferring data we use this data needs to be serialized, which means that any object has to be converted into equivalent text. So, we use XML or JSON. And also, for transferring textual data, we use compression. The storage engines can either be Berkeley DB, which is optimized for small files, or a traditional MySQL database on the store.

(Refer Slide Time: 15:40)

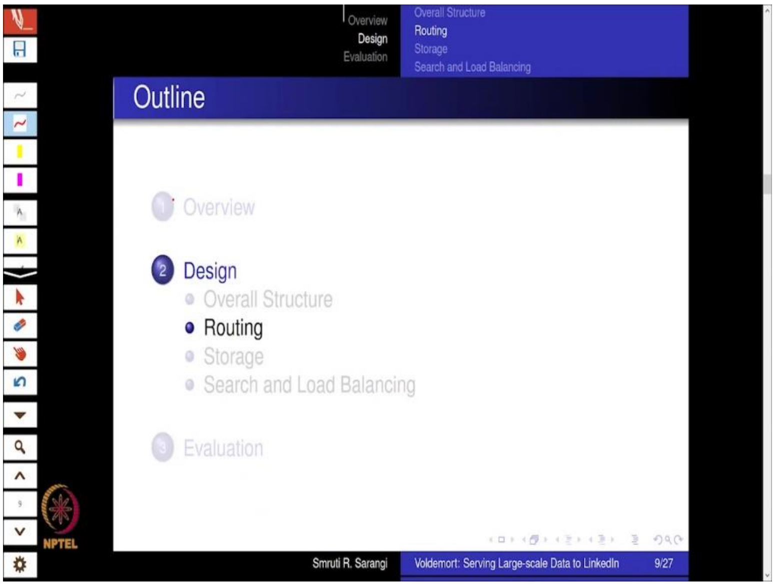


Responses and Requests

So, the idea here is also similar to Dynamo and that we have a client API. So, how does this work? Well so we have these Voldemort clusters within LinkedIn. And the client is not really the browser, but the client is essentially also a LinkedIn node, which is rendering a page for me.

So, the client API would essentially send a request to another network client or a network server, which would access the storage engine, return all the versions, serialize and return as versions. And of course, the clients need to do a conflict resolution, if there are multiple versions. So, since conflict resolution we discussed in great detail in the lectures on CODA and Dynamo. So, I am not going over it.

(Refer Slide Time: 16:29)



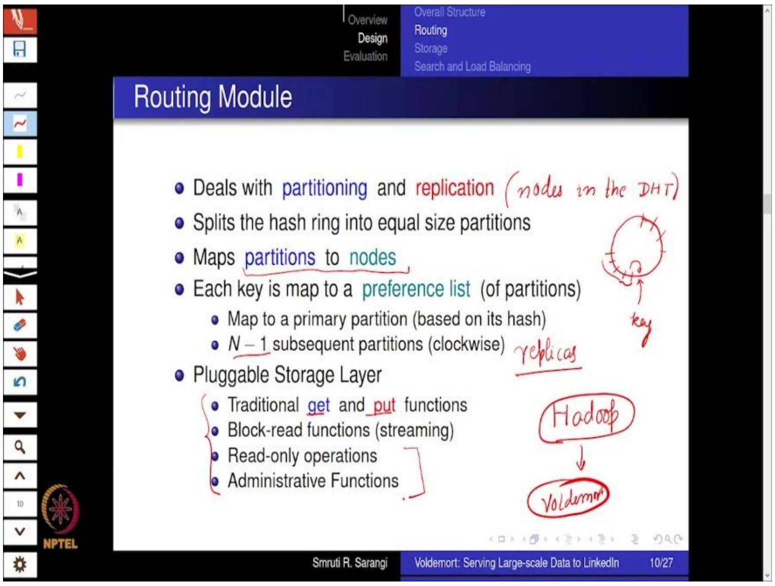
Overview
Design
Evaluation

Overall Structure
Routing
Storage
Search and Load Balancing

Outline

1. Overview
2. Design
   - Overall Structure
   - Routing
   - Storage
   - Search and Load Balancing
3. Evaluation

Smruti R. Sarangi | Voldemort: Serving Large-scale Data to LinkedIn | 9/27



Overview
Design
Evaluation

Overall Structure
Routing
Storage
Search and Load Balancing

Routing Module

- Deals with partitioning and replication (nodes in the DHT)
- Splits the hash ring into equal size partitions
- Maps partitions to nodes
- Each key is map to a preference list (of partitions)
  - Map to a primary partition (based on its hash)
  - $N-1$ subsequent partitions (clockwise) replicas
- Pluggable Storage Layer
  - Traditional get and put functions
  - Block-read functions (streaming)
  - Read-only operations
  - Administrative Functions

Smruti R. Sarangi | Voldemort: Serving Large-scale Data to LinkedIn | 10/27

So, let us now look at the routing. So, the routing module deals with two things partitioning and replication. So, partitioning is the way in which we actually partition the hash space and divide it among the nodes. And the replication is how a given data is replicated among the nodes on the DHT. Let us say in the DHT. So, what we do is that if you would recall, we had a scheme in Dynamo, where we assigned tokens, and it splits the hashing into equal sized partitions.

So, we use exactly the same scheme over here, where we take the hash ring and I may split it into equal sized partitions, then what we do is we map the partitions to the nodes. So, each key, so the moment that, so we have any data, any member ID, or anything, we want to locate on the DHT. Well, we do the same thing. Same idea of hashing, and we map it to a preference list.

So, the preference list will be approved will be a preference list of partitions. So, we can say that look, if the key is mapped over here, on the DHT, we will have a set of partitions, a set of partitions that are kind of like the preference list of the key. So, of course, there will be a primary owner, which can be just the immediate clockwise successor of the key that can be the primary owner of the key, the main partition.

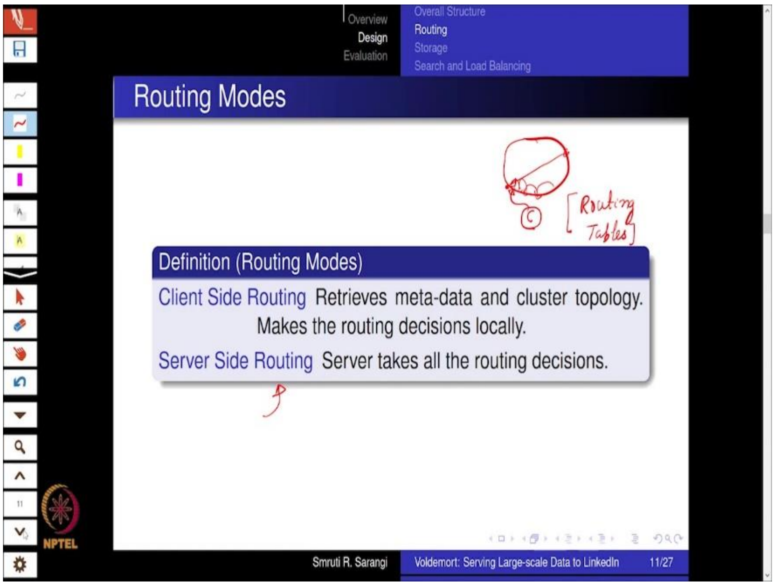
And when we walk kind of clockwise, we will have N - 1 subsequent partition whose main job is to store the replicas. So, this if you would see, this is a very standard architecture that we have been following in this entire lecture series, where we take a DHT we partition it, so this is a very Dynamo like fixed partitioning.

Where a key simply mapped to a partition which is clockwise successor and the N - 1 subsequent partitions subject to the fact that they are on separate physical nodes or we can also say that they are on separate data centers for added reliability. So, this is the broad idea, Dynamo used exactly the same thing. And the storage layer is pluggable. So, we have traditional get input functions.

We also have block read functions in the sense that reading data, we support a streaming read. And the main aim here is that, we need to support primarily read only operations. So, we will see what modifications had to be done to an otherwise vanilla Voldemort system. So, a system is called a vanilla system and when you are using its base version, but in this case, we are not.

So, we are somehow modifying Voldemort to take in data that Hadoop is generating. So, a baseline Hadoop system kind of generates all the data, does all the data mining does everything and kind of bulk loads it into Voldemort, and Voldemort has its DHT which serves as the data. And also, some administrative functions like adding deleting nodes, et cetera, all of them also have to be supported within this setup.

(Refer Slide Time: 20:08)



Routing Modes

Definition (Routing Modes)

Client Side Routing Retrieves meta-data and cluster topology. Makes the routing decisions locally.

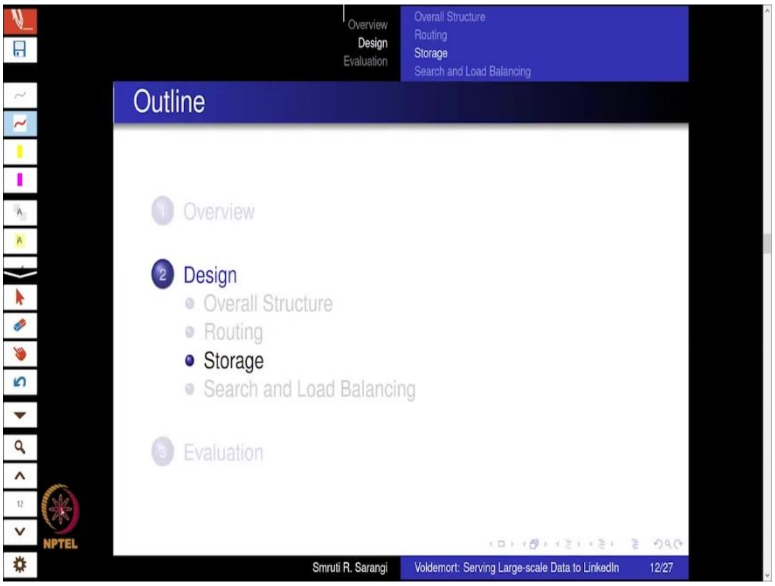
Server Side Routing Server takes all the routing decisions.

So, there are two kinds of routing. So, this was also we are looking at one hop routing. So, here we make a small departure from Amazon Dynamo. So, we support client-side routing, which means that a client is a machine that is lying outside the DHT. So, the client can directly send a message to the partition to the servers or the partition for this specific key, but of course, in that case, the client needs access to all the routing tables.

So, then in that case, rather the what was our traditional routing algorithm, it was that we contact a given server that contacts a few more servers, and then kind of the request jumps, jumps, jumps and reaches the right server. In one hop routing, also, in Dynamo, we contact one server, over here, then it does a single hop insensate over here.

But in client-side routing, the client has all the routing information. The traditional approach is of course server-side routing where the server makes all the routing decisions. And this option is also supported in Voldemort, both are supported, both client side as well as server side.

(Refer Slide Time: 21:37)



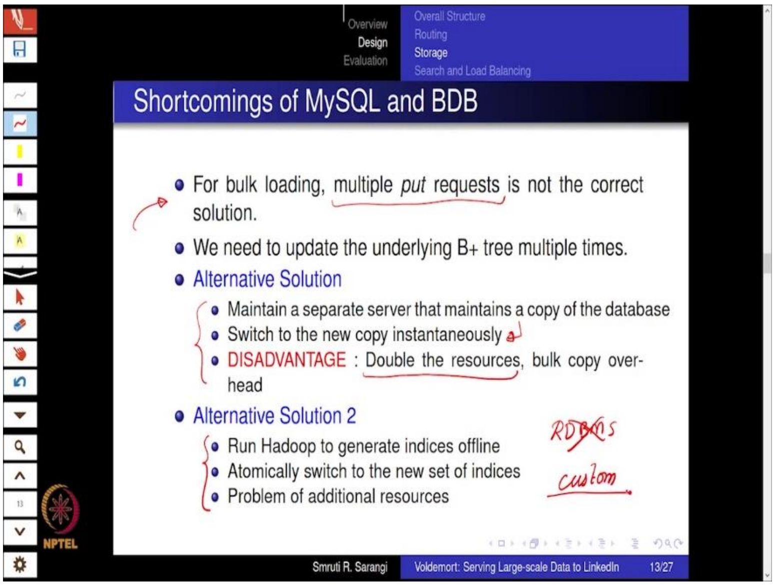
Overview
Design
Evaluation

Overall Structure
Routing
Storage
Search and Load Balancing

Outline

1. Overview
2. Design
   - Overall Structure
   - Routing
   - Storage
   - Search and Load Balancing
3. Evaluation

Smruti R. Sarangi Voldemort: Serving Large-scale Data to LinkedIn 12/27



Overview
Design
Evaluation

Overall Structure
Routing
Storage
Search and Load Balancing

Shortcomings of MySQL and BDB

- For bulk loading, multiple *put* requests is not the correct solution.
- We need to update the underlying B+ tree multiple times.
- Alternative Solution
  - Maintain a separate server that maintains a copy of the database
  - Switch to the new copy instantaneously a
  - DISADVANTAGE : Double the resources, bulk copy overhead
- Alternative Solution 2
  - Run Hadoop to generate indices offline
  - Atomically switch to the new set of indices
  - Problem of additional resources

RDBMS
custom

Smruti R. Sarangi Voldemort: Serving Large-scale Data to LinkedIn 13/27

So, now let us so as far as what we have seen up till now, it is very similar to a traditional vanilla DHT system, client-side routing was new, but most of it is otherwise the same. So, let us look at the storage part. So, first, we will start with the shortcomings of MySQL and Berkeley database.

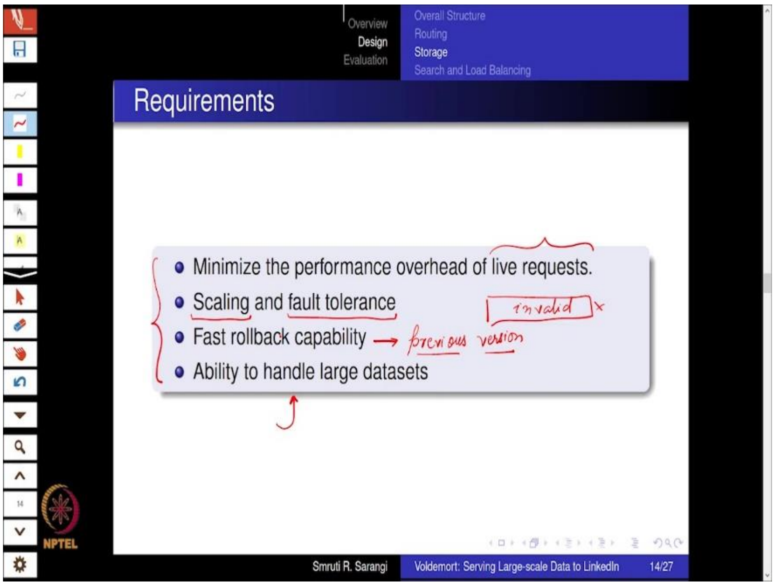
So, here we are talking of bulk loading of data, where a Hadoop cluster kind of all night processes the data that is there in LinkedIn. And once the data is in a nicely processed form, all of it needs to be loaded into Voldemort. So, for this model, having a long sequence of put requests is not the right solution, not the correct solution. So, what? So, the MySQL uses a B plus tree that needs to be updated multiple times, that is rather inefficient.

An alternative solution might be that we maintain a separate server that maintains a copy of the database, we first populate that copy of the database, and we switch to the new copy instantaneously, so we maintain 2 copies. So, this has several disadvantages as well. The first is that we are doubling the number of resources, the amount of resources, there is an overhead of bulk copy of creating the indices, managing the indices, all of that is there.

So, let us look at another alternative solution, which is we run the Hadoop system to generate the indices offline. So, in this case, you have one problem was you generate indices, well, you also generate the indices offline. And then you load in the indices as well. This is fine, it is doable that is maybe the way that many smaller systems actually operate.

But here we need a lot of additional resources and Hadoop along with kind of processing LinkedIn data has to do a lot of work for the relational database. So, what we saw also in the case of Facebook systems is that we will not actually use a relational database management system, this is not something that we will do as a primary source of data. This is not what we will do instead, we will create a custom solution, which is anyway the job of a distributed storage engine.

(Refer Slide Time: 24:14)



Requirements

- Minimize the performance overhead of live requests.
- Scaling and fault tolerance
- Fast rollback capability → previous version
- Ability to handle large datasets

invalid ×

So, what are the requirements? Well, the requirements are that we will minimize the performance overhead of live requests, any requests that we get, regardless of whether the previous Hadoop data is uploading or not, we will minimize the performance overhead, we will of course have fault tolerance, which you always have because of replicas will make the system scale in the sense we will be able to add more servers as and when required.

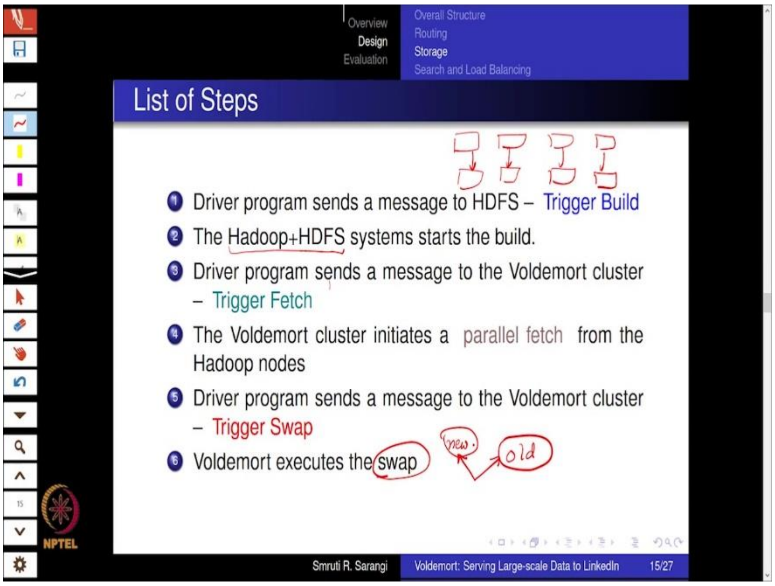
Also, there is a possibility that there might be errors in whatever Hadoop computes. So, it is possible that if Hadoop is giving us a lot of data, the data might actually be invalid. There might be errors and then you know, we might have to jump all of it. So, in this case, we need to have a fast rollback capability to a previous version. That is because for a system as large as LinkedIn, which kind of relies a lot on the relationships that it has computed.

If let us say there is a bug in the code, or if let us say that there is a lot of superfluous data, we do not want to actually end up with a bad state. So, what we would instead like to do is that we would like to roll back to a previous version, where the data might be slightly stale, but it is known to be correct. And if you see most of us also do not update our contact list that frequently, so it will still kind of work.

Also, our system should be able to handle large datasets, say any kind of a Voldemort's system should, in a sense, provide all 4 of these requirements which any distributed system should also ideally provide. And but additional so, scaling fault tolerance, ability to handle large datasets, we have seen that. The two aspects that are kind of new over here is the first the fast rollback capability. In the sense we have not seen that before.

So, for example, we did not see a rollback in let us say Facebook photos for example, but in this case, the entire computed data is very sensitive. So, just in case, we feel that there is some error, we would like to roll back to a safe previous version. And also, we would like to minimize the performance overhead of live requests, because we are never really bulk loaded, read only data, which we are doing right here right now.

(Refer Slide Time: 26:52)



List of Steps

1. Driver program sends a message to HDFS – Trigger Build
2. The Hadoop+HDFS systems starts the build.
3. Driver program sends a message to the Voldemort cluster – Trigger Fetch
4. The Voldemort cluster initiates a parallel fetch from the Hadoop nodes
5. Driver program sends a message to the Voldemort cluster – Trigger Swap
6. Voldemort executes the swap

Smruti R. Sarangi Voldemort: Serving Large-scale Data to LinkedIn 15/27

Let us now go through the exact sequence of steps that typical LinkedIn load would follow. So, the first step is like this. So, the first step is like this, that a driver program would send a message to the HDFS, the Hadoop file system to trigger a build. So, this is the first step. So, the Hadoop plus HDFS systems, they would essentially tree start the build.

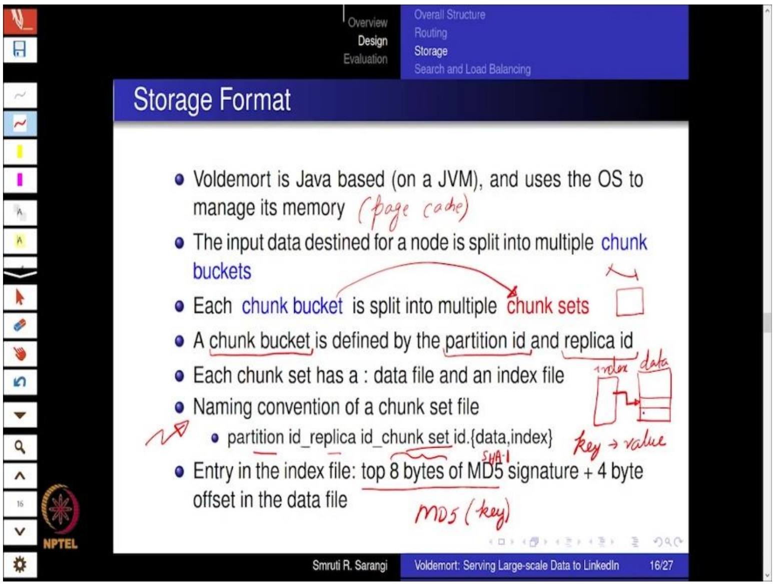
And so, what does it mean? Well, what it means is that they will fetch all the raw data, all the LinkedIn data, they have access to it. So, all the LinkedIn data, the Hadoop system, will take and process and from the data, it will figure out the list of recommendations that you just saw.

It will figure out the list of let us say if I access a certain person's profile, it will also provide the names of other LinkedIn members that whose profile I could go to so on and so forth. So, the driver program sends a message to the Voldemort cluster, asking it to trigger a fetch. The Voldemort clusters a once it is a once the entire data has been generated. The Voldemort cluster initiates a parallel fetch from all the Hadoop nodes. So, pretty much we have a set of distributed Hadoop nodes. And similarly, we have a Voldemort cluster. And essentially, we get a parallel, it triggers a parallel fetch.

So, we have a lot of parallel IO that keeps happening at this time. And then once all the data has been brought into the Voldemort cluster, each of the nodes will actually have 2 versions, a new version and the old version. So, atomically, the driver program will send a message to the Voldemort cluster to trigger a swap and the Voldemort cluster would execute the swap, which means atomically replace it will have a pointer. So, the pointer was, let us say previously proposed pointing to the old version of the data, it will immediately start pointing to the new version of the data, which is what we wanted.

So, it is an instantaneous swap. So, we load the data in the background, we do not have to lock anything there is no disruption in service and instantaneously the pointers switch from the old version to the new version.

(Refer Slide Time: 29:35)



Storage Format

- Voldemort is Java based (on a JVM), and uses the OS to manage its memory (page cache)
- The input data destined for a node is split into multiple chunk buckets
- Each chunk bucket is split into multiple chunk sets
- A chunk bucket is defined by the partition id and replica id
- Each chunk set has a : data file and an index file
- Naming convention of a chunk set file
  - partition id_replica id_chunk set id.{data,index}
- Entry in the index file: top 8 bytes of MD5 signature + 4 byte offset in the data file

The storage format well, so what we discussed? So, we did discuss on a previous slide a storage format that could be used by a MySQL implementation. So, in this case, we are not discussing the MySQL implementation of Voldemort but a custom implementation. So, Voldemort is Java based and it uses the OS, particularly the page cache of the OS to manage its memory. And it assumes that the OS is doing a good job. So, what we do is that we the input data destined for a node is split into multiple chunk buckets. And each chunk bucket is split into multiple chunk sets. So, what is a chunk bucket? Well chunk, for a given partition.

So, of course, for a given partition here is a primary partition not the secondary one. So, for a given partition all the data that maps to a given node, we essentially take all of that and we create a chunk bucket. And of course, we will have multiple such chunk buckets for each replica. So, we can think about this. So, the chunk bucket can be uniquely identified, or can be uniquely named, with the idea of the partition. And the idea of the replica, which replica it is storing.

So, each chunk set. So, within a chunk bucket, of course, we have a chunk sets. And each chunk set, of course stores some data, so we will see what the data is. But in this case, each chunk set is organized similar to Facebook’s photo storage, where we have a data file. And so, data where we have a data file and an index file.

Fair, of course, the spirit of the design is that the index file will be there in memory, and the data file will be there in disk. And the naming convention that we follow over here is that naming convention will be the partition ID, the replica ID. So, partition is essentially a range of the hash space for which, so any key that falls within the range of the hash space, we are storing the data for it. And the replica ID, so which replica is this.

And so, this partition, and replica will essentially pointers to the chunk bucket. And within the chunk bucket, we will of course have chunk sets. So, then we get the chunk set ID. And then of course, the last will be either the data, data file or the index file. So, the entry in the index file will of course be the top 8 bytes or the MD5 signature of the data. And so basically, the index file will have two things.

So, the first set will have two fields, the first field will be that given the data that we are trying to access, what we do is we compute an MD5 signature of its. So, first MD5 is another hashing algorithm, similar to what we have seen similar to SHA1 that chord uses. So, we use the top 8 bytes of it.

And then once we are able to find the index entry, we use a 32 bit offset into the data file, similar to what Facebook’s photo storage used to do. So, what do we do? Well, what we do is that we search through the index file, till we find the entry that we are interested in. This will be the top 8 bytes of the MD5 signature. And from this, we find a 4-byte offset into the data file.

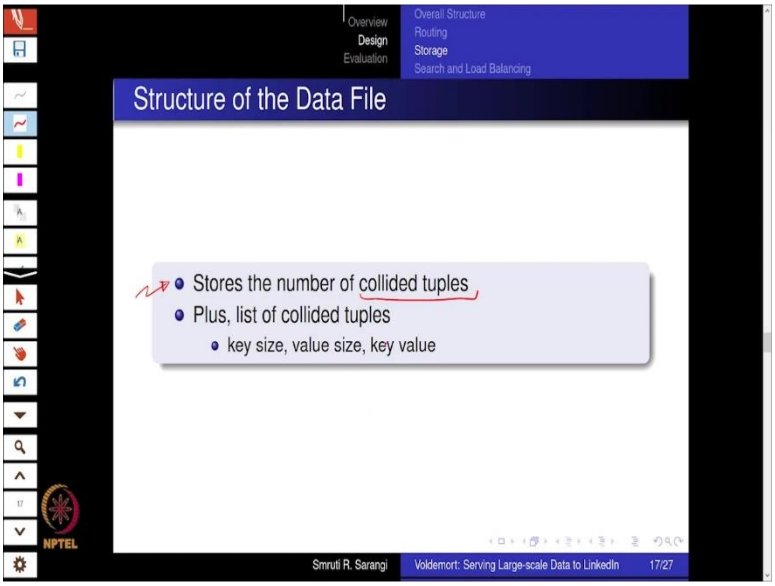
Now, that at this point, I have kept the discussion reasonably independent of LinkedIn. This is because Voldemort is a generic system. And so, till now, I have not introduced any LinkedIn specific things. That is the reason what I have maintained is the 8 bytes MD5 signature.

And, of course, this is the MD5 signature of the key. So, previously, what we were doing is that we were taking the data of the key and computing in SHA1, instead of SHA1 that we have used in all the DHT that we have seen up till now. We use another hashing algorithm, which does the same thing.

So, what does the DHT do? What does the DHT does is given a key it gives us the value. How do we do that? Well, we compute the hash of a key and we find the node that stores it. How did we compute the hash? Well, we use the SHA1 signature, instead of SHA1. In this case, we just use MD5 that does the same thing with MD5 has some better properties with regards to uniformity. That is the reason it is used. And of course, we use the top 8 bytes of if not all the bytes. And then we locate this on the index file inside a given chunk set, of course, and that points us to the data file. so, the main aim over here is that given a key, we would like to go to the right partition.

And then that is the first thing and we would essentially route ourselves to the node that contains the partition. Within the partition, we will, for a given replica ID, we will try to find the chunk set ID. And in the chunk set ID, again, we will have an index file and a data file. And here we will use, we will not use SHA1 signature, but the MD5 signature to essentially locate ourselves in the index file. And then there will be a pointer into the data file.

(Refer Slide Time: 35:34)

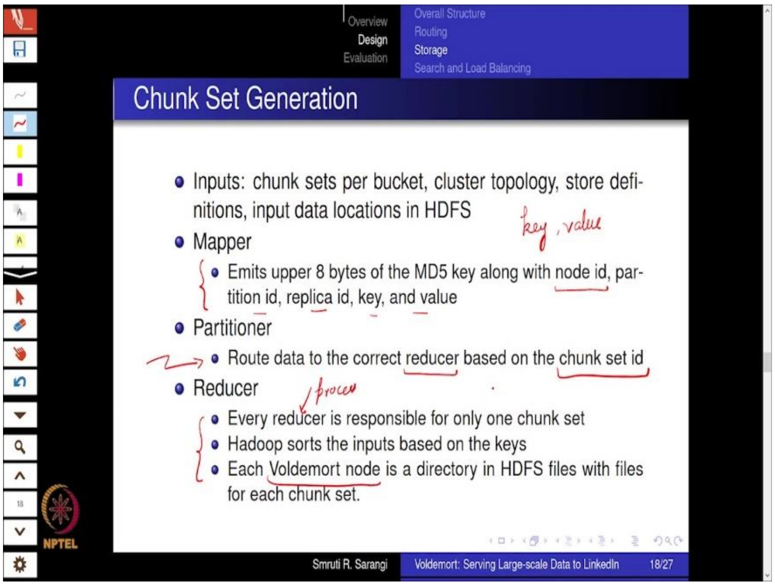


Structure of the Data File

- Stores the number of collided tuples
- Plus, list of collided tuples
  - key size, value size, key value

So, it is possible that we can have hash collisions in the data file. So, we will have a number of collided tuples. So, the data file additionally will store a list of collided tuples that will store the number of key value pairs that have collided with a given key value. And the list of collided tuples will be stored at the end of the data file.

(Refer Slide Time: 36:00)



Overview
Design
Evaluation

Overall Structure
Routing
Storage
Search and Load Balancing

Chunk Set Generation

- Inputs: chunk sets per bucket, cluster topology, store definitions, input data locations in HDFS
- Mapper
  - Emits upper 8 bytes of the MD5 key along with node id, partition id, replica id, key, and value
- Partitioner
  - Route data to the correct reducer based on the chunk set id
- Reducer
  - Every reducer is responsible for only one chunk set
  - Hadoop sorts the inputs based on the keys
  - Each Voldemort node is a directory in HDFS files with files for each chunk set.

key, value

process

Smruti R. Sarangi Voldemort: Serving Large-scale Data to LinkedIn 18/27

So, how do we generate chunk sets? Well, so that is the main idea that the input will be the chunk sets per bucket, the cluster topology, the definitions of where the stores are into HDFS. So, we will have a mapper that maps that emits the upper 8 bytes. So, the MD5 key along with some more data. So, this will be the node ID, the partition ID, the replica ID, the key and the value.

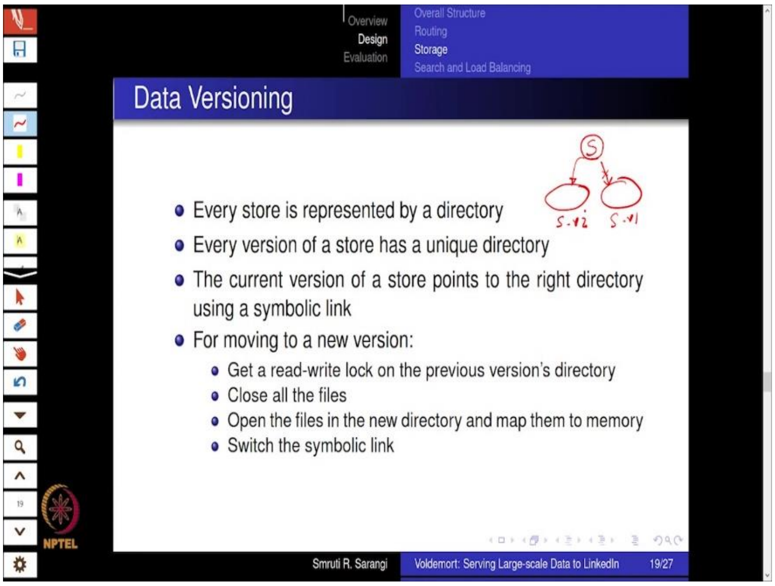
So, the job of the matter is that when the HDFS system is generating data, it will generate data in key value pairs. So, we will discuss the exact semantics of what is the key and what is the value in the context of LinkedIn in latest slides. But for the time being, let us assume that it is a key value pair.

So, what we do is that the job of the mapper is to take a key and a value computer 75 Key and map it to a given node partition. So, essentially map it to a set of replicas. So, first map it to a node. And then from the node, we will find the partition that is the on and off, then we will find a set of replicas, store them in those replica servers for that particular key value.

So, the partitioner, what does it do? Well, it routes data to the correct reducer, based on a chunk set ID, what is a reducer? A reducer further processes the data. So, recall that we are doing a MapReduce kind of computation. So, for every chunk set ID we first extract all the relevant raw data from HDFS, then what the reducer does is that it processes all of that data and stores it in the node ID that we have computed in the past. Say every reducer, let us call the reducer a dedicated process is responsible for only one chunk set.

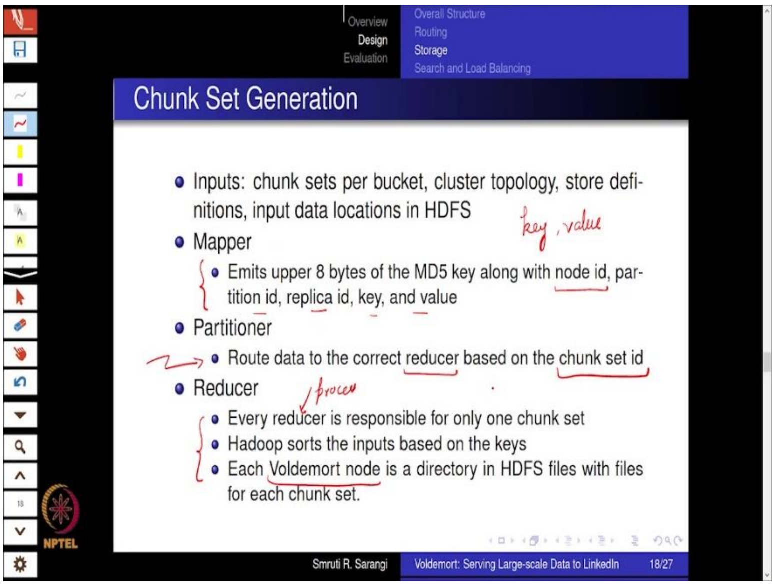
So, what Hadoop would do is it would sort the inputs, all the inputs based on the keys, and each Voldemort data, each Voldemort node, each node is Voldemort, is a directory in the baseline HDFS file system. And we have a separate file for every chunk set. So, essentially, for every chunk set, we have a separate file, and we generate that in Hadoop. And that contains all the key value pairs that we would be interested in.

(Refer Slide Time: 38:42)



Data Versioning

- Every store is represented by a directory
- Every version of a store has a unique directory
- The current version of a store points to the right directory using a symbolic link
- For moving to a new version:
  - Get a read-write lock on the previous version's directory
  - Close all the files
  - Open the files in the new directory and map them to memory
  - Switch the symbolic link



Chunk Set Generation

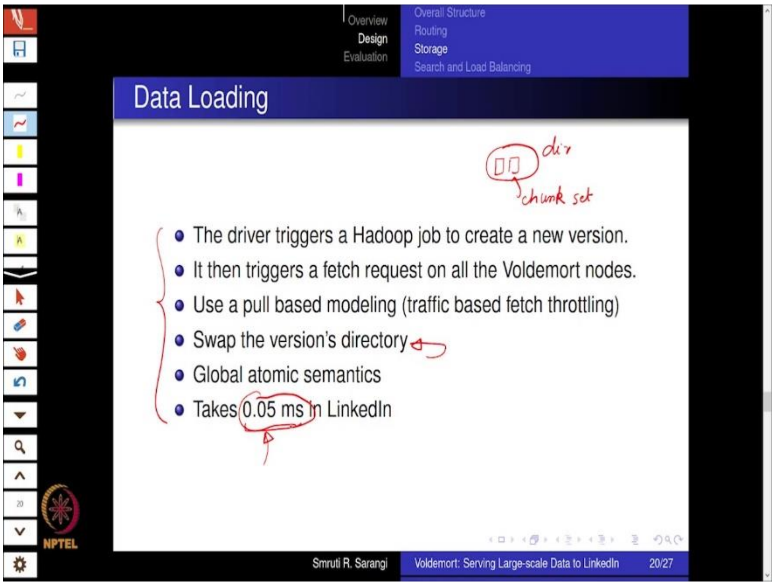
- Inputs: chunk sets per bucket, cluster topology, store definitions, input data locations in HDFS
- Mapper
  - Emits upper 8 bytes of the MD5 key along with node id, partition id, replica id, key, and value
- Partitioner
  - Route data to the correct reducer based on the chunk set id
- Reducer
  - Every reducer is responsible for only one chunk set
  - Hadoop sorts the inputs based on the keys
  - Each Voldemort node is a directory in HDFS files with files for each chunk set.

So, given that given store is represented by a directory, which we saw in the last sentence over here, that every store, every Voldemort node, which is essentially a store is represented by a directory. This allows us to store versions very easily in the sense that every version of a store will have a unique directory.

So, we can say that let us say for a given store, this is version 1, for the given store this is version 2. So, the current version of a store would point to the right directory using a symbolic link. So, which is very easy. So, let us say that for given store S that will just be a symbolic link in the file system. It will previously be pointing to version 1 will make it stop pointing to version 1, will make it start pointing to version 2.

So, for moving to a new version, what we will do is we will get a read write lock on the previous version's directory, will close all the files, will open new files in a new directory, map them to memory and then switch to symbolic link. And so, basically it is important that when we switch the symbolic link for the new directory, all the files are already map to memory.

(Refer Slide Time: 40:08)



Overview
Design
Evaluation

Overall Structure
Routing
Storage
Search and Load Balancing

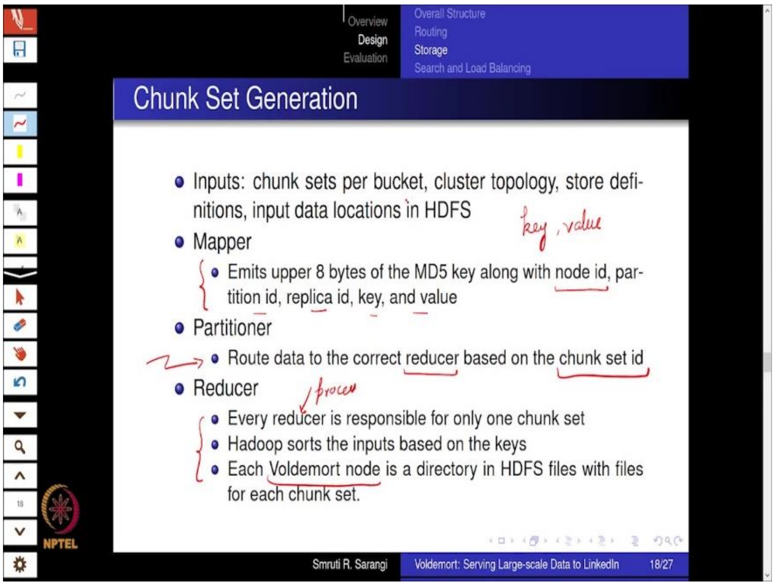
Data Loading

dir

chunk set

- The driver triggers a Hadoop job to create a new version.
- It then triggers a fetch request on all the Voldemort nodes.
- Use a pull based modeling (traffic based fetch throttling)
- Swap the version's directory
- Global atomic semantics
- Takes 0.05 ms in LinkedIn

Smruti R. Sarangi Voldemort: Serving Large-scale Data to LinkedIn 20/27



Overview
Design
Evaluation

Overall Structure
Routing
Storage
Search and Load Balancing

Chunk Set Generation

- Inputs: chunk sets per bucket, cluster topology, store definitions, input data locations in HDFS
- Mapper
  - Emits upper 8 bytes of the MD5 key along with node id, partition id, replica id, key, and value
- Partitioner
  - Route data to the correct reducer based on the chunk set id
- Reducer
  - Every reducer is responsible for only one chunk set
  - Hadoop sorts the inputs based on the keys
  - Each Voldemort node is a directory in HDFS files with files for each chunk set.

key, value

process

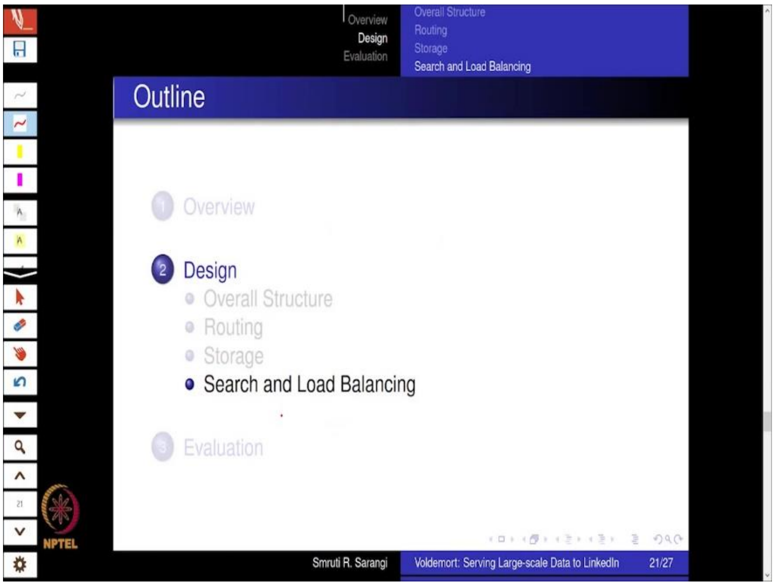
Smruti R. Sarangi Voldemort: Serving Large-scale Data to LinkedIn 18/27

Data Loading, how does it work? Well, that driver program that we have that triggers a Hadoop job to create a new version. Once the new version has been created, so what does this mean? Well, the new version over here would essentially create different chunk sets per chunk bucket. Once all of that has been created, and pretty much for every chunk bucket, we store it in a directory, in the directory or multiple files, where each file is a chunk set.

Once we have done that, the Voldemort nodes pull these directories into the local file system. And then atomically, do a swap as we are mentioned. So, in the version of LinkedIn that corresponded to this paper, this was a very, very fast operation.

And so, which was this the swap process, this was a very fast operation, it took around 0.05 milliseconds. So, which is 50 microseconds to actually do the swap. And this happened, atomical.

(Refer Slide Time: 41:19)



Overview
Design
Evaluation

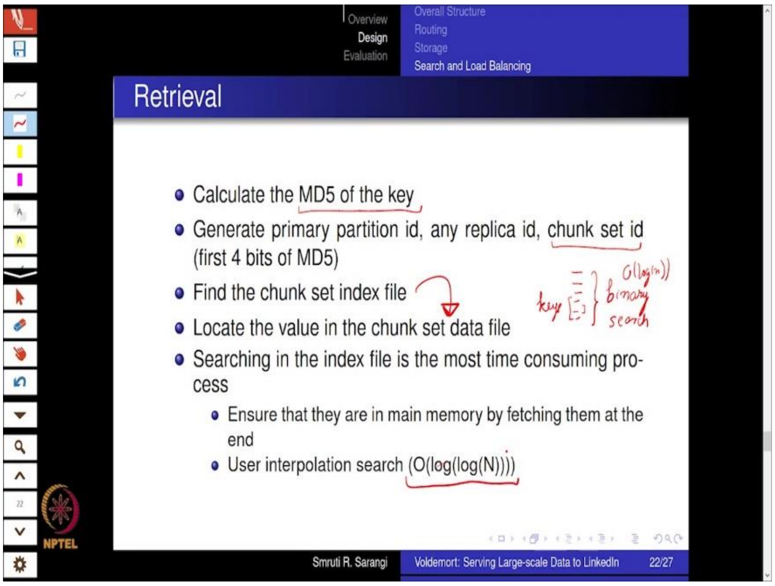
Overall Structure
Routing
Storage
Search and Load Balancing

Outline

1. Overview
2. Design
   - Overall Structure
   - Routing
   - Storage
   - Search and Load Balancing
3. Evaluation

Smruti R. Sarangi

Voldemort: Serving Large-scale Data to LinkedIn 21/27



Retrieval

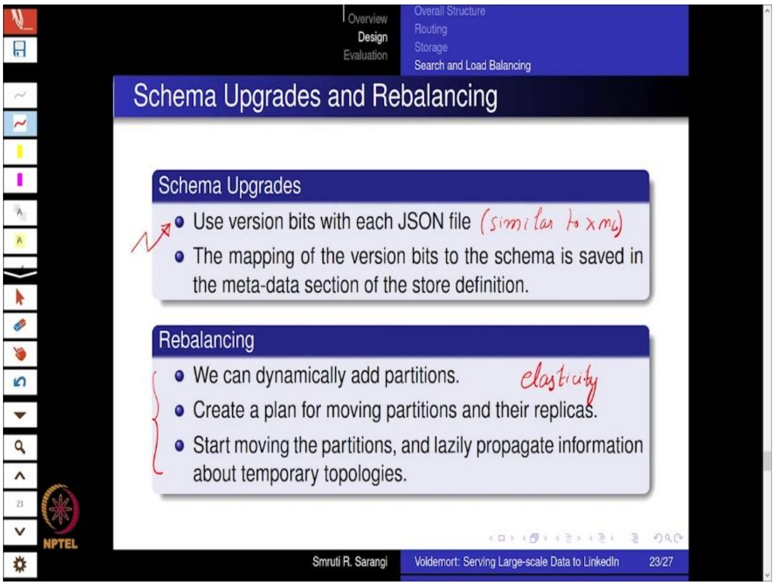
- Calculate the MD5 of the key
- Generate primary partition id, any replica id, chunk set id (first 4 bits of MD5)
- Find the chunk set index file
- Locate the value in the chunk set data file
- Searching in the index file is the most time consuming process
  - Ensure that they are in main memory by fetching them at the end
  - User interpolation search (O(log(log(N))))

Now, a little bit of search and load balancing. So, the retrieval goes like this, that we calculate the MD5 hash of the key. And from that, we generate the ID of the primary partition. If we are going to a replica, then we can find the ID of the replica. And the idea of the chunk set, which are set to be the first 4 bits of MD5 key from that we access the machine on the DHT and we find the junk set index file.

From the chunk set index file, we locate the values within the chunk set data file. So, how do we do that? So, in an index file, let us say that based on the bits from the MD5 hash, all the keys are stored. So, one option is that we do a binary search. So, if we do a binary search, it will take us all around login time. This further might involve multiple discrete is a part of this is stored on disk, what we can instead do is we can perform an interpolation search, which means that we take a look at the keys.

And based on that since they are in sorted order, we make an approximate guess of maybe the keys lie in this range. Once we go to that range, we again make an approximate guess and keep on trying to find (())(42:54) in the range. So, interpolation search is faster in the sense, it is faster than binary search is O(log(log(N)))). But of course, the search is probabilistic. In the sense it is a probabilistic random algorithm, which kind of relies on an expected distribution of the keys.

(Refer Slide Time: 43:15)



Schema Upgrades and Rebalancing

Schema Upgrades

- Use version bits with each JSON file (similar to xml)
- The mapping of the version bits to the schema is saved in the meta-data section of the store definition.

Rebalancing

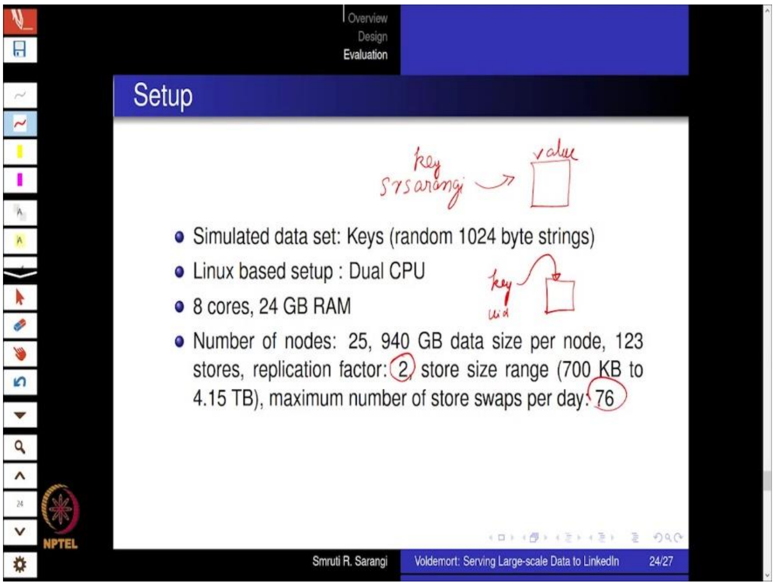
- We can dynamically add partitions. elasticity
- Create a plan for moving partitions and their replicas.
- Start moving the partitions, and lazily propagate information about temporary topologies.

Now, for rebalancing and schema upgrades so well, we might want to change the schema of the chunks or the chunk sets. So, to do that, we will actually use version bits with every JSON file. So, JSON is the textual as is the text format is similar to XML. It is a text format in which data gets transferred between Hadoop and Voldemort.

So, every JSON file will have some version bits to indicate the specific schema that it is using in this meta data. And also, for rebalancing to handle any additional pressure on the nodes. We can dynamically add partitions, we can create a plan for moving partitions and the replicas to new nodes.

And also, we can start moving the partitions and lazily propagate information about the temporary topologies. So, that also can be done. And so, all of this kind of support the elasticity of the server farm, in the sense that as we keep on adding more servers, our system gradually keeps on growing.

(Refer Slide Time: 44:29)



Overview
Design
Evaluation

Setup

- Simulated data set: Keys (random 1024 byte strings)
- Linux based setup : Dual CPU
- 8 cores, 24 GB RAM
- Number of nodes: 25, 940 GB data size per node, 123 stores, replication factor: 2, store size range (700 KB to 4.15 TB), maximum number of store swaps per day: 76

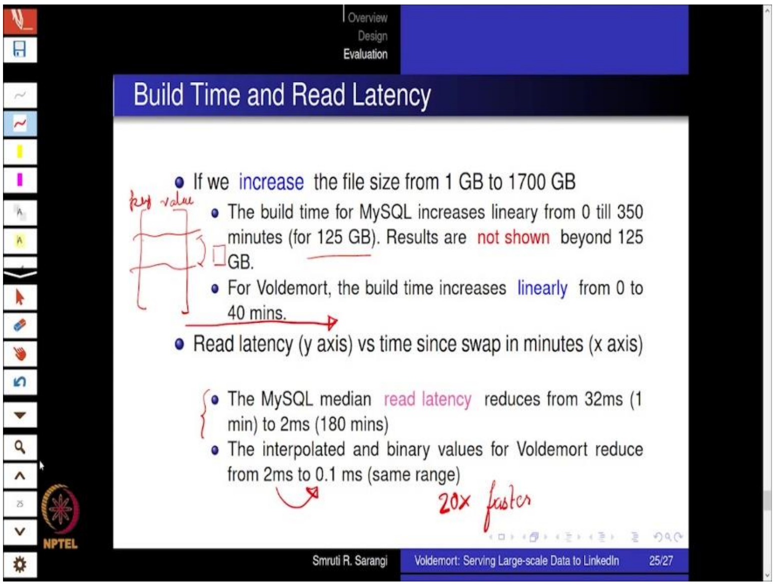
Smruti R. Sarangi | Voldemort: Serving Large-scale Data to LinkedIn | 24/27

So, the experimental setup, well, we had a simulated data set. The keys were random 1024-byte strings. So, in an actual LinkedIn system, it may not be this way, the key can actually be my member ID. So, my member ID, Srsarangi, that can be my LinkedIn key. So, what Voldemort would do is if it would like to fetch all the recommendations, so the value would be all the people that I know, all the people that I should know, all the LinkedIn recommendations, this can be the key, and this can be the value.

Alternatively, let us see if you visit my profile. So, then the key again will be my user ID, and the profile will be see and let us say on the side, when you get the list of people that have also visited my profile, that could be the value. But for the sake of this experiment, the keys were just random 1024-byte strings.

Baseline Linux system was used with 8 cores, 24 GB RAM. And the number of nodes was set to 25, with roughly 940 Giga Bytes data size per node, 123 stores any replication factor of 2. In a sense, each data item was replicated on 2 servers. The size of the store data did vary, it had a very wide variance from 700 kilobytes to 4.15 terabytes. And the maximum number of store swapes per day was set to 76.

(Refer Slide Time: 46:06)



Build Time and Read Latency

- If we increase the file size from 1 GB to 1700 GB
  - The build time for MySQL increases lineary from 0 till 350 minutes (for 125 GB). Results are not shown beyond 125 GB.
  - For Voldemort, the build time increases linearly from 0 to 40 mins.
- Read latency (y axis) vs time since swap in minutes (x axis)
  - The MySQL median read latency reduces from 32ms (1 min) to 2ms (180 mins)
  - The interpolated and binary values for Voldemort reduce from 2ms to 0.1 ms (same range)

So, what was observed is that the building time as we increase the file size from 1 GB to 1700 GB. If let us say I were to use a MySQL database, the build time for MySQL would increase linearly from 0 till 350 minutes or 125 gigabytes. Henceforth, the overhead of storing something in MySQL, in terms of calculating the index structures and so on is so high, that it becomes an order of magnitude slower than Voldemort.

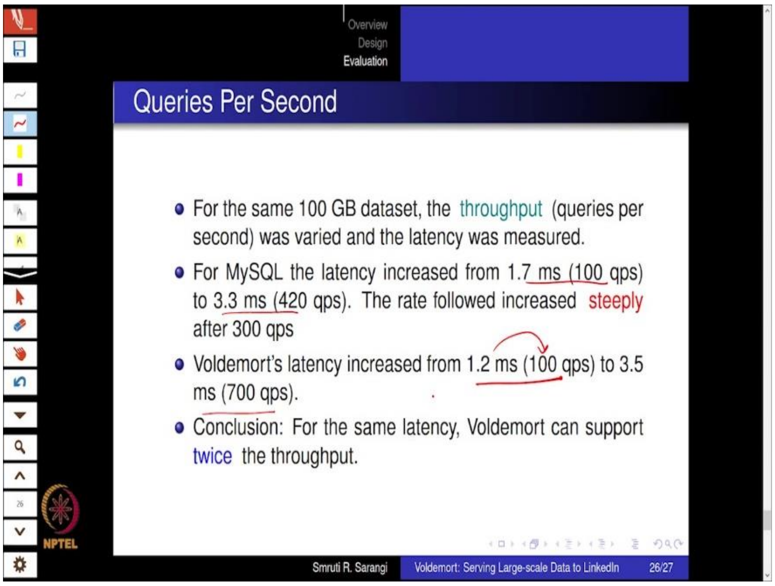
But for Voldemort, even with 1700 gigabytes, within 40 minutes, the entire data can be generated as well as stored in Voldemort’s format. So, of course, when we are comparing between MySQL and Voldemort, the inherent build time is the same. But to bring it to my SQL’s format, which means to create the indexing data structures, it actually takes a very long time. In Voldemort, we do not need that much of time, because essentially, we are partitioning all this.

So, let us look at it in a different way. If we are creating a billion key value pairs, and you are storing them in MySQL. MySQL will have to create a very large indexing structure, which takes time. In this case, what we are doing is we are kind of partitioning the keys based on the hash of the key, we are doing a hash space partition.

And for every partition, we are essentially treating that as a chunk set, and we are creating a small file for it within the store. This is much faster. So, we will see Voldemort’s internal system is much faster. The Read latency versus the time swap well, for MySQL, the median read latency reduces from 30 milliseconds to 2 milliseconds, mainly because most of this data comes into the cache.

For both interpolated and binary searching these values for Voldemort reduced from 2 milliseconds to 0.1 millisecond. So, Voldemort is still within the same range still 20X Faster, which is expected. Given that it has the small index structures for the small data files, instead of one large index structure for the entire database. So, the 20X speed up here is kind of expected.

(Refer Slide Time: 48:43)



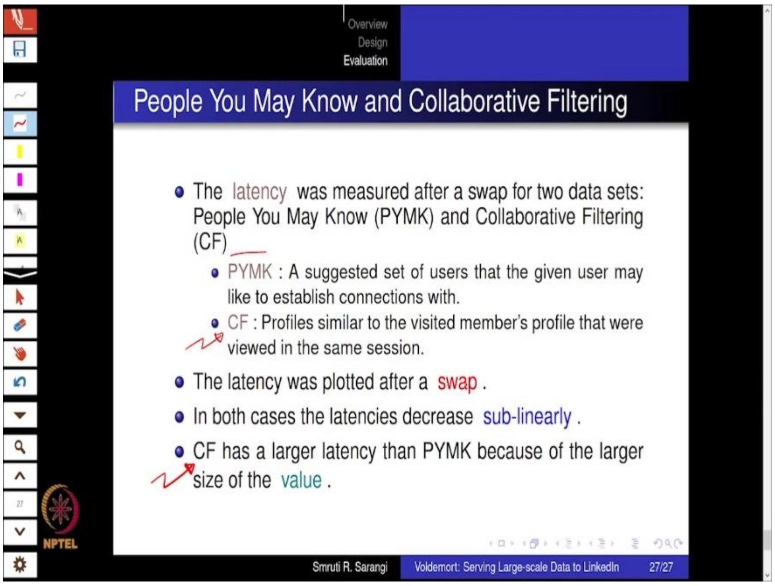
Queries Per Second

- For the same 100 GB dataset, the throughput (queries per second) was varied and the latency was measured.
- For MySQL the latency increased from 1.7 ms (100 qps) to 3.3 ms (420 qps). The rate followed increased steeply after 300 qps
- Voldemort's latency increased from 1.2 ms (100 qps) to 3.5 ms (700 qps).
- Conclusion: For the same latency, Voldemort can support twice the throughput.

And if I were to compare the qps of the queries per second, for the same 100 gigabyte data set, the throughput was varied and the latency was measured. For MySQL, the latency decreased for 100 qps, it was 1.7 milliseconds. For 420 qps, it was 3, roughly 3.3 milliseconds.

In comparison, Voldemort was latency increased far more gradually. So, it will we were able to scale it, the authors were able to scale it from 1.2 millisecond for 100 qps. Just compare the numbers over here and here to 3.5 millisecond and 700 qps. Just compare these 2 lines, which means that for the same latency, Voldemort can support roughly twice the throughput.

(Refer Slide Time: 49:36)



Overview
Design
Evaluation

People You May Know and Collaborative Filtering

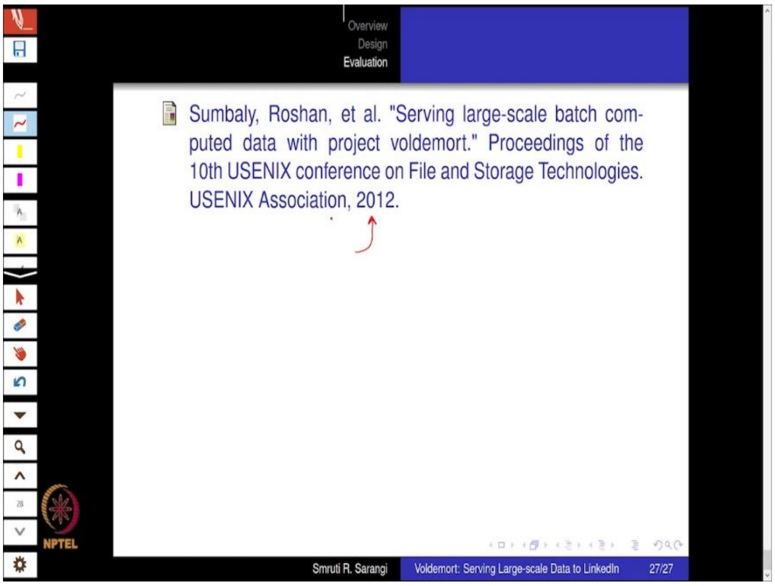
- The latency was measured after a swap for two data sets: People You May Know (PYMK) and Collaborative Filtering (CF)
  - PYMK : A suggested set of users that the given user may like to establish connections with.
  - CF : Profiles similar to the visited member's profile that were viewed in the same session.
- The latency was plotted after a swap .
- In both cases the latencies decrease sub-linearly .
- CF has a larger latency than PYMK because of the larger size of the value .

Smruti R. Sarangi   Voldemort: Serving Large-scale Data to LinkedIn   27/27

Then so, this the previous experiment was on randomly generated keys and data. In this case, it was done more with a LinkedIn specific data set. The first is people you may know, which is something that you saw when I showed you the demo, which is a suggested set of users may like to establish a connection with and other was collaborative filtering, which was profile similar to the visited members profile.

The latency was plotted after a swap. In both cases, the latency is decreased sub linearly. And CF has a larger latency than PYMK, because of the larger size of the value, but of course, this depends on how many contacts are shown. And on the LinkedIn version. Nowadays, of course, the people you may know, has increased to a very long list. But so, this can be this is kind of an idiosyncrasy of the specific LinkedIn version. But the important point is that both of them, both of these operations were really fast.

(Refer Slide Time: 50:45)



Sumbaly, Roshan, et al. "Serving large-scale batch computed data with project voldemort." Proceedings of the 10th USENIX conference on File and Storage Technologies. USENIX Association, 2012.

So, this brings us to an end of this lecture. This paper was published in 2012. And it has been 8 years ever since. So, in these 8 years, the number of LinkedIn users have clearly increased by leaps and bounds. So, we need to wait for more information in terms of research papers to understand how exactly the increase in scale has been handled.