DAAD Summerschool Curitiba 2011

Aspects of Large Scale High Speed Computing Building Blocks of a Cloud

Storage Networks

6: PAST: Peer-to-Peer Network Storage

Christian Schindelhauer
Technical Faculty
Computer-Networks and Telematics
University of Freiburg

PAST

- PAST: A large-scale, persistent peer-to-peer storage utility
  - by Peter Druschel (Rice University, Houston – now Max-Planck-Institut, Saarbrücken/Kaiserlautern)
  - and Antony Rowstron (Microsoft Research)
- Literature
  - A. Rowstron and P. Druschel, "Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility", 18th ACM SOSP'01, 2001.
    - all pictures from this paper
  - P. Druschel and A. Rowstron, "PAST: A large-scale, persistent peer-to-peer storage utility", HotOS VIII, May 2001.

Goals of PAST

- Peer-to-Peer based Internet Storage
  - on top of Pastry
- Goals
  - File based storage
  - High availability of data
  - Persistent storage
  - Scalability
  - Efficient usage of resources

Motivation

- Multiple, diverse nodes in the Internet can be used
  - safety by different locations
- No complicated backup
  - No additional backup devices
  - No mirroring
  - No RAID or SAN systems with special hardware
- Joint use of storage
  - for sharing files
  - for publishing documents
- Overcome local storage and data safety limitations

Interface of PAST

- Create:
  `fileId = Insert(name, owner-credentials, k, file)`
  - stores a file at a user-specified number k of divers nodes within the PAST network
  - produces a 160 bit ID which identifies the file (via SHA-1)
- Lookup:
  `file = Lookup(fileId)`
  - reliably retrieves a copy of the file identified fileId
- Reclaim:
  `Reclaim(fileId, owner-credentials)`
  - reclaims the storage occupied by the k copies of the file identified by fileId

- Other operations do not exist:
  - No erase
    - to avoid complex agreement protocols
  - No write or rename
    - to avoid write conflicts
  - No group right management
    - to avoid user, group managements
  - No list files, file information, etc.
- Such operations must be provided by additional layer

Relevant Parts of Pastry

- Leafset:
  - Neighbors on the ring
- Routing Table
  - Nodes for each prefix + 1 other letter
- Neighborhood set
  - set of nodes which have small TTL

NodeId 10233102

Leaf set

SMALLER		LARGER	
10233033	10233021	10233120	10233122
10233001	10233000	10233230	10233232

Routing table

-0-2212102	**1**	-2-2301203	-3-1203203
0	1-1-301233	1-2-230203	1-3-021022
10-0-31203	10-1-32102	**2**	10-3-23302
102-0-0230	102-1-1302	102-2-2302	**3**
1023-0-322	1023-1-000	1023-2-121	**3**
10233-0-01	**1**	10233-2-32	
0		102331-2-0	
		2	

Neighborhood set

13021022	10200230	11301233	31301233
02212102	22301203	31203203	33213321

Interfaces of Pastry

- route(M, X):
  - route message M to node with nodeId numerically closest to X
- deliver(M):
  - deliver message M to application
- forwarding(M, X):
  - message M is being forwarded towards key X
- newLeaf(L):
  - report change in leaf set L to application

Insert Request Operation

- Compute fileId by hashing
  - file name
  - public key of client
  - some random numbers, called salt
- Storage (k x filesize)
  - is debited against client‘s quota
- File certificate
  - is produced and signed with owner‘s private key
  - contains fileID, SHA-1 hash of file‘s content, replciation factor k, the random salt, creation date, etc.
- File and certificate are routed via Pastry
  - to node responsible for fileID
- When it arrives in one node of the k nodes close to the fileId
  - the node checks the validityof the file
  - it is duplicated to all other k-1 nodes numerically close to fileId
- When all k nodes have accepted a copy
  - Each nodes sends store receipt is send to the owner
- If something goes wrong an error message is sent back
  - and nothing stored

Lookup

- Client sends message with requested fileId into the Pastry network
- The first node storing the file answers
  - no further routing
- The node sends back the file
- Locality property of Pastry helps to send a close-by copy of a file

Reclaim

- Client‘s nodes sends reclaim certificate
  - allowing the storing nodes to check that the claim is authentificated
- Each node sends a reclaim receipt
- The client sends this recept to the retrieve the storage from the quota management

Security

- Smartcard
  - for PAST users which want to store files
  - generates and verifies all certificates
  - maintain the storage quotas
  - ensure the integrity of nodeID and fileID assignment
- Users/nodes without smartcard
  - can read and serve as storage servers
- Randomized routing
  - prevents intersection of messages
- Malicious nodes only have local influence

Storage Management

- Goals
  - Utilization of all storage
  - Storage balancing
  - Providing k file replicas
- Methods
  - Replica diversion
    - exception to storing replicas nodes in the leafset
  - File diversion
    - if the local nodes are full all replicas are stored at different locations

Causes of Storage Load Imbalance

- Statistical variation
  - birthday paradoxon (on a weaker scale)
- High variance of the size distribution
  - Typical heavy-tail distribution, e.g. Pareto distribution
- Different storage capacity of PAST nodes

Per-Node Storage

- Assumption:
  - Storage of nodes differ by at most a factor of 100
- Large scale storage
  - must be inserted as multiple PAST nodes
- Storage control:
  - if a node storage is too large it is asked to split and rejoin
  - if a node storage is too small it is rejected

Replica Diversion

- **The first node close to the fileld checks whether it can store the file**
  - if yes, it does and sends the store receipt
- **If a node A cannot store the file, it tries replica diversion**
  - A chooses a node B in its leaf set which is not among the k closest asks B to store the copy
  - If B accepts, A stores a pointer to B and sends a store receipt
- **When A or B fails then the replica is inaccessible**
  - failure probability is doubled

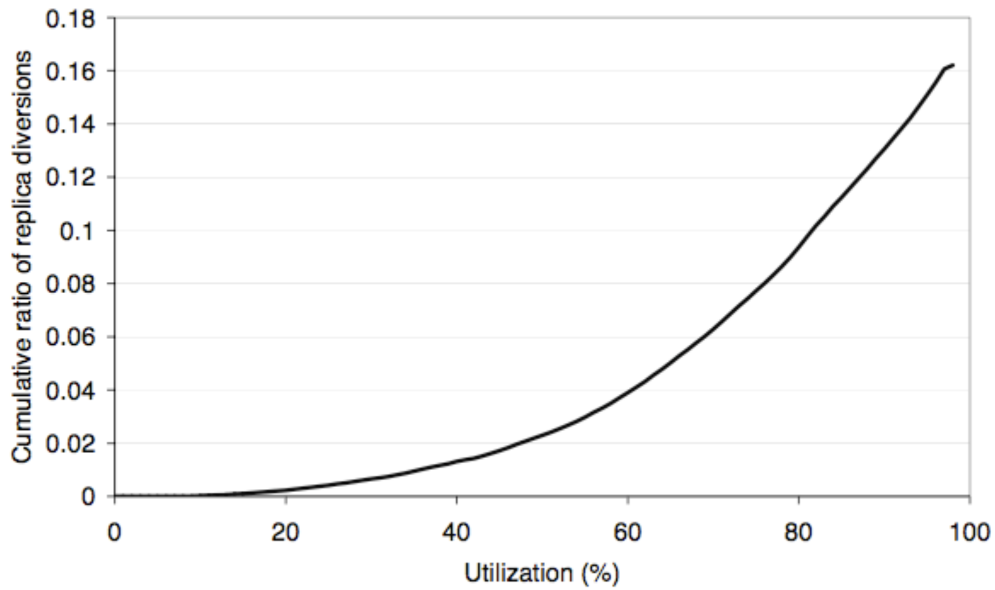


Figure 5: Cumulative ratio of replica diversions versus storage utilization, when $t_{pri} = 0.1$ and $t_{div} = 0.05$.

Policies for Replica Diversion

- Acceptance of replicas at a node
  - If (size of a file)/(remaining free space) > t then reject the file
    - for different t`s for close nodes (t_{pri}) and far nodes (t_{div}), where $t_{pri} > t_{div}$
  - discriminates large files and far storage
- Selecting a node to store a diverted replica
  - in the leaf set and
  - not in the k nodes closest to the fileId
  - do not hold a diverted replica of the same file
- Deciding when to divert a file to different part of the Pastry ring
  - If one of the k nodes does not find a proxy node
  - then it sends a reject message
  - and all nodes for the replicas discard the file

File Diversion

- If k nodes close to the chosen fileId
  - cannot store the file
  - nor divert the replicas locally in the leafset
- then an error message is sent to the client
- The client generates a new fileId using different sal
  - and repeats the insert operation up to 3 times
  - then the operation is aborted and a failure is reported to the application
- Possibly the application retries with small fragments of the file

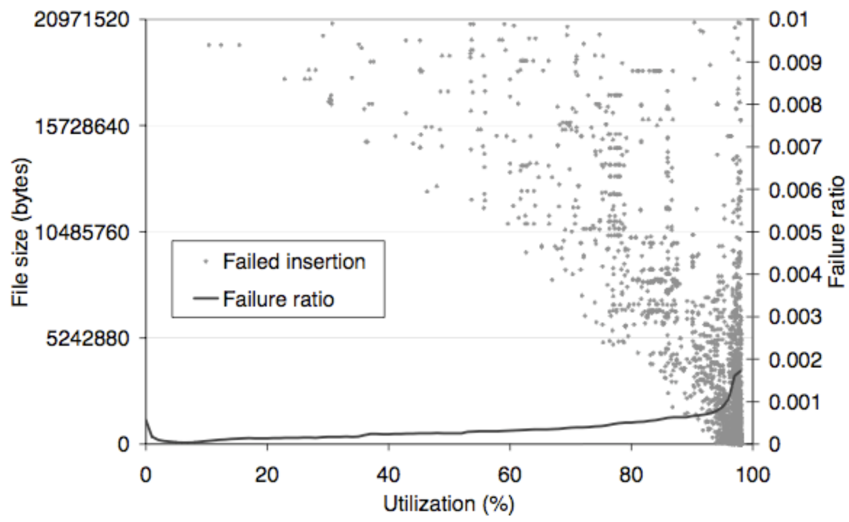


Figure 7: File insertion failures versus storage utilization for the filesystem workload, when $t_{pri} = 0.1$, $t_{div} = 0.05$.

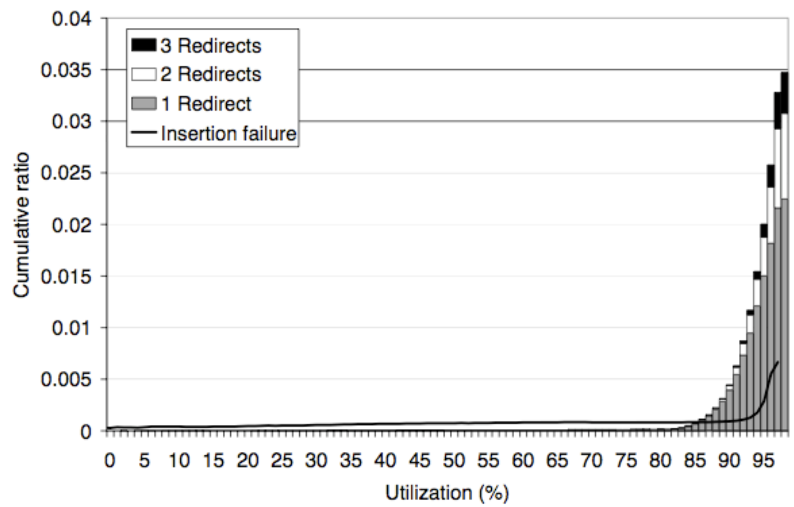


Figure 4: Ratio of file diversions and cumulative insertion failures versus storage utilization, $t_{pri} = 0.1$ and $t_{div} = 0.05$.

Maintaining Replicas

- Pastry protocols checks leaf set periodically
- Node failure has been recognized
    - if a node is unresponsive for some certain time
  - Pastry triggers adjustment of the leaf set
    - PAST redistributes replicas
  - if the new neighbor is too full, then other nodes in the nodes will be uses via replica diversion
- When a new node arrives
  - files are not moved, but pointers adjusted (replica diversion)
  - because of ratio of storage to bandwidth

File Encoding

- k replicas is not the best redundancy strategy
- Using a Reed-Solomon encoding
  - with m additional check sum blocks to n original data blocks
  - reduces the storage overhead to (m+n)/n times the file size
    - if all m+n shares are distributed over different nodes
  - possibly speeds upt the access spee
- PAST
  - does NOT use any such encoding techniques

Caching

- Goal:
  - Minimize fetch distance
  - Maximize query throughput
  - Balance the query load
- Replicas provide these features
  - Highly popular files may demand many more replicas
    - this is provided by cache management
- PAST nodes use „unused“ portion to cache files
  - cached copies can be erased at any time
    - e.g. for storing primary of redirected replicas
- When a file is routed through a node during lookup or insert it is inserted into the local cache
- Cache replacement policy: GreedyDual-Size
  - considers aging, file size and costs of a file

Experimental Results Caching

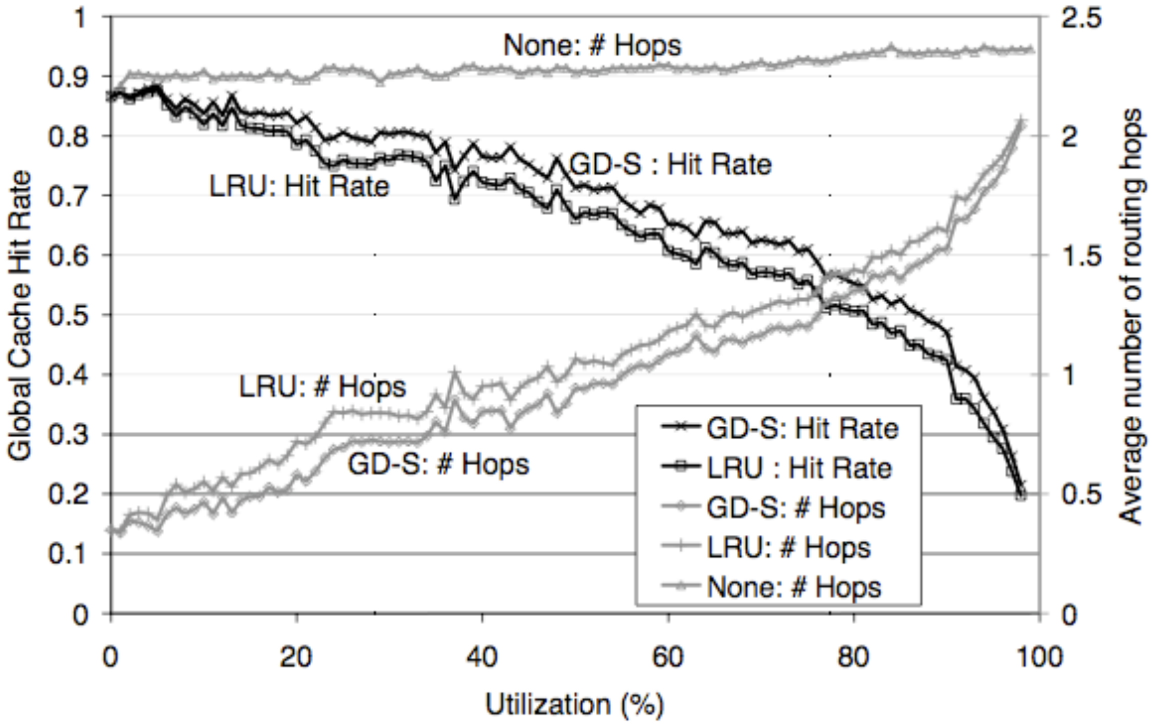


Figure 8: Global cache hit ratio and average number of message hops versus utilization using Least-Recently-Used (LRU), GreedyDual-Size (GD-S), and no caching, with $t_{pri} = 0.1$ and $t_{div} = 0.05$.

Summary

- PAST provides a distributed storage system
  - which allows full storage usage and locality features
- Storage management
  - based ond Smartcard system
    - provides a hardware restriction
  - utilization moderately increases failure rates and time behavior

DAAD Summerschool Curitiba 2011

Aspects of Large Scale High Speed Computing Building Blocks of a Cloud

Storage Networks

6: Optimizing Heterogeneous Data Distribution

Christian Schindelhauer
Technical Faculty
Computer-Networks and Telematics
University of Freiburg

Literature

- André Brinkmann, Kay Salzwedel, Christian Scheideler, Compact, Adaptive Placement Schemes for Non-Uniform Capacities, 14th ACM Symposium on Parallelism in Algorithms and Architectures 2002 (SPAA 2002)
- Christian Schindelhauer, Gunnar Schomaker, Weighted Distributed Hash Tables, 17th ACM Symposium on Parallelism in Algorithms and Architectures 2005 (SPAA 2005)
- Christian Schindelhauer, Gunnar Schomaker, SAN Optimal Multi Parameter Access Scheme, ICN 2006, International Conference on Networking, Mauritius, April 23-26, 2006

The Problem in Storage Networks

CoNe Freiburg

- $A_{d,s}$: Number of bytes of document d assigned to storage s
- Distributed Algorithm:
  - Use DHHT to split each document into |S| parts
  - Store corresponding blocks on the server
- Can be also achieved by a centralized algorithm
- Straight forward generalization of fair balance
  - Distribute data according to a (m x n) distribution matrix A where
    $\forall s : \sum_d A_{d,s} \leq |s|$ and $\forall d : \sum_s A_{d,s} = |d|$
- DHHT
  - assigns $A_{d,s}(1 \pm \varepsilon)$ elements of d ∈ D to s ∈ S
  - Information needed: File-IDs, Server-IDs, and matrix A
  - If matrix A changes to A´ $(1+\varepsilon)\sum_{d,s} \left| A_{d,s} - A'_{d,s} \right|$
    data reassignments are needed

UNI FREIBURG

How to Balance

- A fair balance like $A_{d,s} = |d| \cdot \frac{|s|}{\sum_{s' \in S} |s'|}$ is not always the best to do
- Servers are different in capacity and bandwidth
- Documents are different in size and popularity

- Goal: Optimize Time

- Assumption
  - All sizes can be modeled as real numbers

Which Time ?

- b(s) = bandwidth of server s
  - b(s) = number of bytes per second
- p(d) = popularity of document d
  - p(d) = number of read/write accesses
- Sequential time for a document d and an assignment A

$$\text{SeqTime}_A(d) := \sum_{s \in S} \frac{A_{d,s}}{b(s)}$$

- Parallel time for a document d and an assignment A

$$\text{ParTime}_A(d) := \max_{s \in S} \left\{ \frac{A_{d,s}}{b(s)} \right\}$$

- Observation
  - Popular bytes cause more traffic than less popular once
  - Costs are defined by the traffic per byte

Sequential Time

- Sequential time
  - load all parts of a document from all servers sequentially

$$\text{SeqTime}_A(d) := \sum_{s \in \mathcal{S}} \frac{A_{d,s}}{b(s)}$$

- Worst case sequential time

$$\text{WSeqTime} := \max_d \{\text{SeqTime}_A(d)\}$$

- Average sequential time

$$\text{AvSeqTime} := \sum_{d \in \mathcal{D}} p(d) \ \text{SeqTime}_A(d)$$

- where
  - S: set of servers with bandwidth b(s) and capacity |s| for each server s
  - D: set of documents with size |d| and popularity p(d) for each document

Parallel Time

- Parallel time
  - load all parts of a document from all servers simultaneously

$$\text{ParTime}_A(d) := \max_{s \in \mathcal{S}} \left\{ \frac{A_{d,s}}{b(s)} \right\}$$

- Worst case parallel time

$$\text{WParTime} := \max_d \{\text{ParTime}_A(d)\}$$

- Average parallel time

$$\text{AvParTime} := \sum_{d \in \mathcal{D}} p(d) \ \text{ParTime}_A(d)$$

- where
  - S: set of servers with bandwidth b(s) and capacity |s| for each server s
  - D: set of documents with size |d| and popularity p(d) for each document

Sequential Bandwidth

- Sequential time
  - load all parts of a document from all servers sequentially

$$\mathrm{SeqTime}_A(d) := \sum_{s \in \mathcal{S}} \frac{A_{d,s}}{b(s)}$$

- Sequential bandwidth
  - download speed of a document d

$$\mathrm{SeqBandwidth}_A(d) := \frac{|d|}{\mathrm{SeqTime}_A(d)}$$

- Worst case sequential bandwidth

  WBandwidth := mind {SeqBandwidthA(d)}

- Average sequential bandwidth

  AvBandwidth := $\sum_{d \in \mathcal{D}} p(d)$ SeqBandwidth(d)

- where
  - S: set of servers with bandwidth b(s) and capacity |s| for each server s
  - D: set of documents with size |d| and popularity p(d) for each document

Parallel Bandwidth

CoNe Freiburg

- Parallel time
  - load all parts of a document from all servers in parallel

$$\mathrm{ParTime}_A(d) := \max_{s \in \mathcal{S}} \left\{ \frac{A_{d,s}}{b(s)} \right\}$$

- Parallel bandwidth
  - download speed of a datum d

$$\mathrm{ParBandwidth}_A(d) := \frac{|d|}{\mathrm{ParTime}_A(d)}$$

- Worst case parallel bandwidth

$$\mathrm{WParBandwidth} := \min_d \{\mathrm{ParBandwidth}_A(d)\}$$

- Average parallel bandwidth time

$$\mathrm{AvParBandwidth} := \sum_{d \in \mathcal{D}} p(d)\ \mathrm{ParBandwidth}_A(d)$$

- where
  - S: set of servers with bandwidth b(s) and capacity |s| for each server s
  - D: set of documents with size |d| and popularity p(d) for each document

UNI FREIBURG

Most Reasonable Time Measures

- Minimize the expected sequential time based on popularity of the document:

$$\text{AvSeqTime}(p, A) = \sum_{d \in \mathcal{D}} \sum_{s \in \mathcal{S}} p(d) \frac{A_{d,s}}{b(s)}$$

- Minimize the expected parallel time based on the popularity of the document

$$\text{AvParTime}(p, A) = \sum_{d \in D} \max_{s \in S} \frac{A_{d,s}}{b(s)} p(d)$$

Solution by Linear Program

$$\forall s : \sum_d A_{d,s} \leq |s| \qquad \forall d : \sum_s A_{d,s} = |d|$$

Measure	Linear programm	Add. variables	Additional restraint	Optimize
AvSeqTime	yes	—	—	$\min \sum_{s\in\mathcal{S}} \sum_{d\in\mathcal{D}} p(d) \frac{A_{d,s}}{b(s)}$
WSeqTime	yes	m	$\forall d \in \mathcal{D} : \sum_{s\in\mathcal{S}} \frac{A_{d,s}}{b(s)} \leq m$	$\min m$
AvParTime	yes	$(m_d)_{d\in\mathcal{D}}$	$\forall s \in \mathcal{S}, \forall d \in \mathcal{D} : \frac{A_{d,s}}{b(s)} \leq m_d$	$\min \sum_{d\in\mathcal{D}} p(d) m_d$
WParTime	yes	m	$\forall s \in \mathcal{S}, \forall d \in \mathcal{D} : \frac{A_{d,s}}{b(s)} \leq m$	$\min M$
AvSeqBandwidth	no	—	—	$\max \sum_{d\in\mathcal{D}} \frac{p(d)\|d\|}{\sum_{s\in\mathcal{S}} \frac{A_{d,s}}{b(s)}}$
WSeqBandwidth	yes	m	$\forall d \in \mathcal{D} : \sum_{s\in\mathcal{S}} \frac{A_{d,s}}{\|d\| b(s)} \leq m$	$\min m$
AvParBandwidth	no	$(m_d)_{d\in\mathcal{D}}$	$\forall d \in \mathcal{D} : \sum_{s\in\mathcal{S}} \frac{A_{d,s}}{b(s)\|d\|} \leq m_d$	$\max \sum_{d\in\mathcal{D}} \frac{p(d)}{m_d}$
WParBandwidth	yes	m	$\forall s \in \mathcal{S}, \forall d \in \mathcal{D} : \frac{A_{d,s}}{\|d\| b(s)} \leq m$	$\min m$

How to Describe AvParTime as a LP

AvParTime

$$= \sum_{d \in D} p(d) \cdot \underbrace{\max_{s \in S} \frac{A_{d,s}}{b(s)}}_{m_d}$$

$$= \sum_{d \in D} p(d) \cdot m_d$$

Variables: $A_{d,s}$, m_d

Restraints: $\sum_s A_{d,s} = |d|$

$\sum_d A_{d,s} \leq |s|$

$$m_d = \max_{s \in S} \frac{A_{d,s}}{b(s)}$$

Additional Restraints

$$m_d \geq \frac{1}{b(s_1)} \cdot A_{d,s_1}$$

$$m_d \geq \frac{1}{b(s_2)} \cdot A_{d,s_2}$$

$\vdots$

CoNe Freiburg

UNI FREIBURG

Example

- **Storage device**
  - s_1: 500 GB, 100 MB/s
  - s_2: 100 GB, 50 MB/s
  - s_3: 1 GB 1000 MB/s
- **Documents**
  - d_1: 100 GB, popularity 1/111
  - d_2: 5 GB, popularity 100/111
  - d_3: 100 GB, popularity 10/111

$A_{d,s}$	s_1	s_2	s_3	Σ
d_1	100	0	0	100
d_2	2	2	1	5
d_3	2	98	0	100
Σ	≤ 500	≤ 100	≤ 1	

	SeqTime	SeqBand width	ParTime	ParBand width
d_1	1000	100	1000	100
d_2	61	82	40	125
d_3	1980	51	1960	51
Av	1864	121	1827	160
Worst case	1980	51	1960	51

Excursion: Linear Programming

- Linear Program (Linear Optimization)
- Given: m × n matrix A

  m-dimensional vector b

  n-dimensional vector c
- Find: n-dimensional vector $x=(x_1, ..., x_n)$
- such that
  - $x \geq 0$, i.e. for all j: $x_j \geq 0$
  - A x = b, i. e. $$\sum_{j=1}^{n} \sum_{i=1}^{m} A_{ij} x_j = b_j$$
  - $z = c^T x$ is minimized, i.e. $z = \sum_{j=1}^{n} c_j x_j$ is minimal

Linear Programming 2

- Linear Programming (LP2)
- Given: m × n matrix A

  m-dimensional vector b

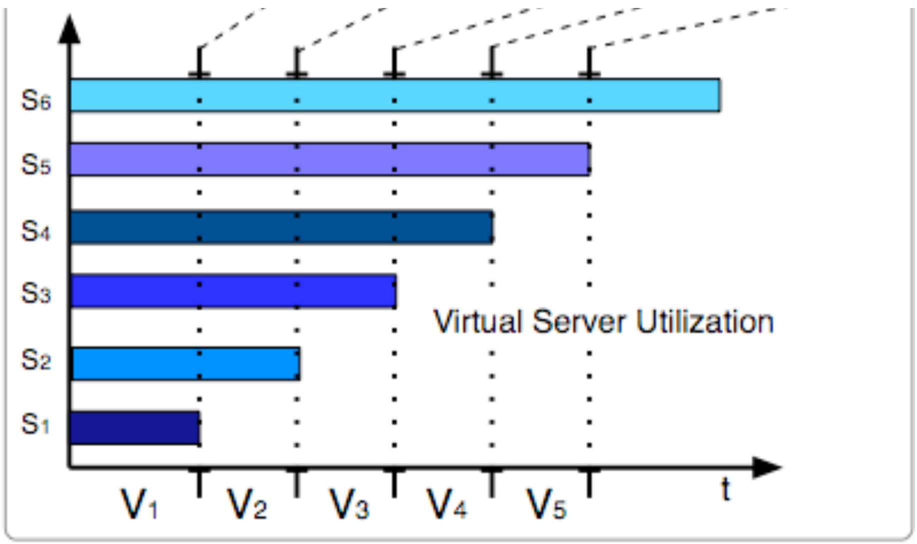
  n-dimensional vector c
- Find: n-dimensional vector $x=(x_1, ..., x_n)$
- such that
  - $x \geq 0$
  - $A x \leq b$
  - $z = c^T x$ is maximal

Performance of Linear Programming

- Worst case time behavior of the Simplex algorithm is exponential
  - A simplex can have an exponential number of edges
- For randomized inputs, the running time of Simplex is polynomial on the expectation
- The Ellipsoid algorithm is a different method with polynomial worst case behavior
  - In practice it is usually outperformed by the Simplex algorithm

ParTime = SeqTime with virtual servers

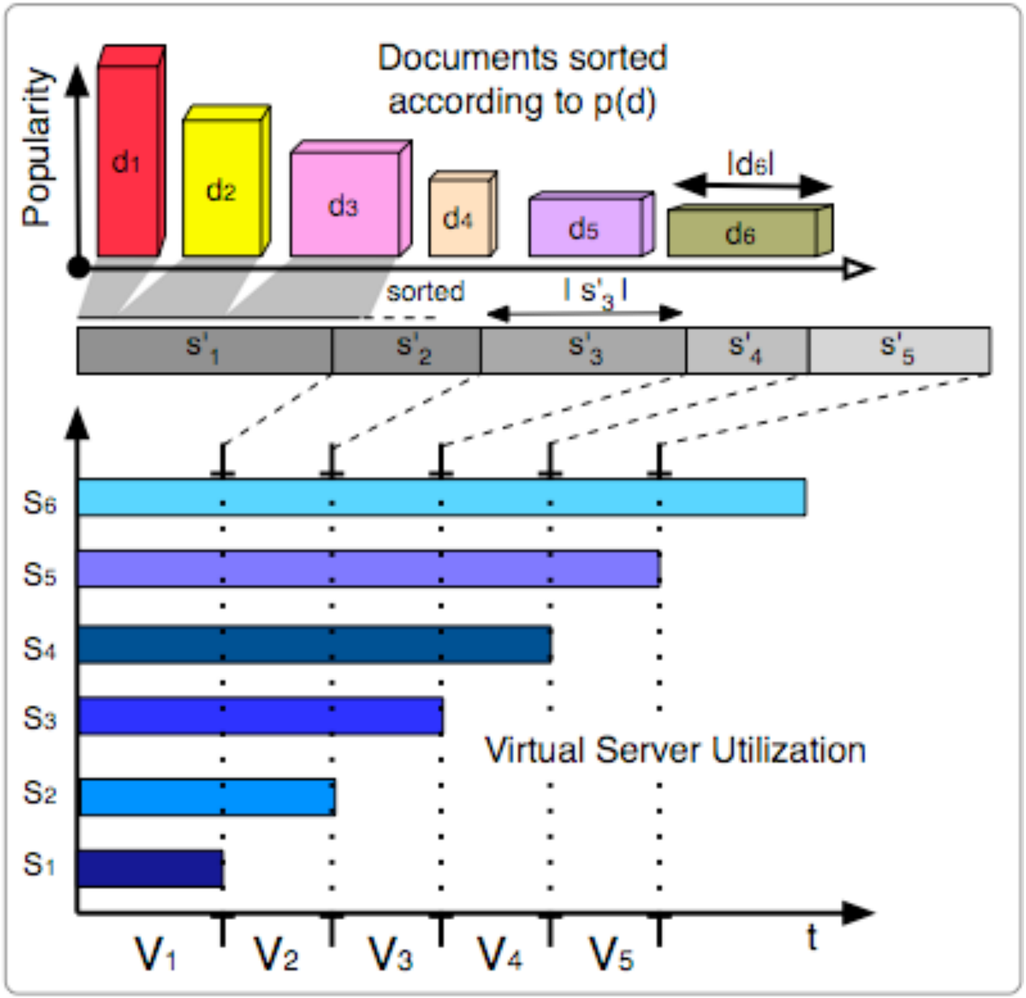
- **Reduce optimal solution for LP of ParTime to the optimal solution of LP of SeqTime**
  - Combining capacity of many disks in parallel
- **Define new sequential virtual servers $s'_1, \ldots, s'_m$**
  - Sort s_i such that $\frac{|s_j|}{b(s_j)} \leq \frac{|s_{j+1}|}{b(s_{j+1})}$
  - Server s'_j parallelizes servers $s_j, \ldots, s_{|S|}$
  - Virtual servers s'_i are then sorted such that $b(s'_i) > b(s'_{i+1})$
  - Size of s'_i:



$$t_j = \frac{|s_j|}{b(s_j)} - \sum_{i=1}^{j-1} t_i \qquad s'_j = b(s'_j) \cdot t_j$$

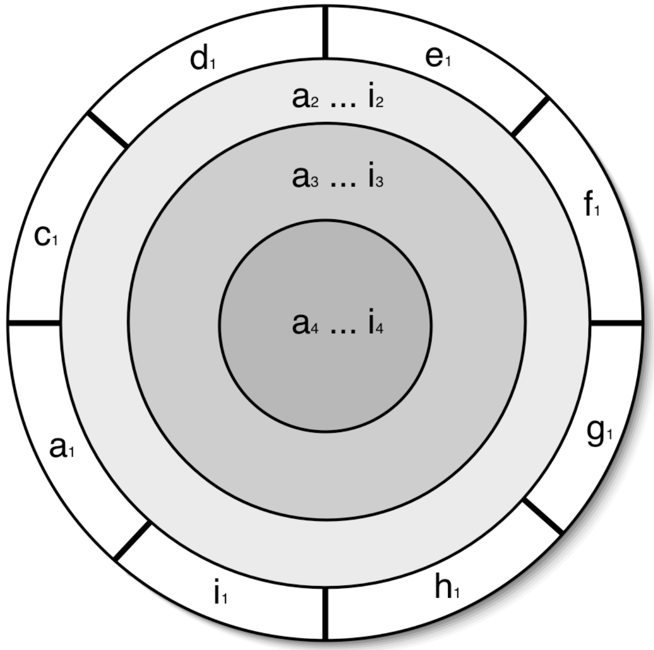
Solve the LP of AvSeqTime

- **Simple optimal greedy solution**
- **Repeat until all documents are assinged:**
  - Assign most popular document on fastest sequential (virtual) server
  - Reduce the storage of the server by the document size and remove the document



Applications in Storage Networks

- **Object storage with different popularity zones**
  - e.g. movies with varying popularities over time
  - Fragmentation is done automatically
  - Includes dynamics for adding and removing documents
  - The same for servers



- **Use different bandwidth**
  - Each disk has different bandwidths
  - Exporting different zone classes as sequential servers

DAAD Summerschool Curitiba 2011

Aspects of Large Scale High Speed Computing Building Blocks of a Cloud

Storage Networks

6: Optimizing Heterogeneous Data Distribution

Christian Schindelhauer
Technical Faculty
Computer-Networks and Telematics
University of Freiburg