Storage Tradeoffs in a Collaborative Backup Service for Mobile Devices

Ludovic Courtès, Marc-Olivier Killijian, David Powell

20 October 2006
Context

The MoSAIC Project

- **3-year project** started in Sept. 2004: IRISA, Eurecom and LAAS-CNRS
- supported by the French national program for Security and Informatics (ACI S&I)

Target

- **communicating mobile devices** (laptops, PDAs, cell phones)
- **mobile ad-hoc networks**, spontaneous, peer-to-peer-like interactions

Dependability Goals

- improving **data availability**
- guarantee **data integrity & confidentiality**
Goals and Issues
- Fault Tolerance for Mobile Devices
- Challenges

Storage Mechanisms
Preliminary Evaluation of Storage Mechanisms
Fault Tolerance for Mobile Devices

Costly and Complex Backup

- only **intermittent access** to one’s desktop machine
- potentially **costly communications** (e.g., GPRS, UMTS)

Our Approach: **Cooperative Backup** (illustrated)

- leverage encounters, **opportunistically**
- high **throughput**, low **energetic cost** (Wifi, Bluetooth, etc.)
- leverage **excess resources**
- variety of **independent failure modes**
- hopefully **self-managed** mechanism
Challenges

Secure Cooperation

- participants have no *a priori* trust relationship
- protect against **DoS attacks**: data retention, selfishness, flooding
- ideas from P2P: *reputation* mechanism, *cooperation incentives*, etc.

Trustworthy Data Storage

- ensure data **confidentiality**
- data **integrity**
- data **authenticity**
- more requirements...
Goals and Issues

**Storage Mechanisms**
- Constraints Imposed on the Storage Layer
- Maximizing Storage Efficiency
- Chopping Data Into Small Blocks
- Providing a Suitable Meta-Data Format
- Providing Data Confidentiality, Integrity, and Authenticity
- Enforcing Backup Atomicity
- Replication Using Erasure Codes

Preliminary Evaluation of Storage Mechanisms
Constraints Imposed on the Storage Layer

**Scarce Resources** (energy, storage, CPU)

- maximize **storage efficiency**
- but **avoid CPU-intensive techniques** (compression, encryption)

**Short-lived and Unpredictable Encounters**

- fragment data into small blocks & **disseminate** it among contributors
- yet, **retain transactional semantics** of the backup (ACID)

**Lack of Trust Among Participants**

- replicate data fragments
- enforce data **confidentiality**, verify **integrity & authenticity**
Maximizing Storage Efficiency

Single-Instance Storage

⇒ reduce redundancy across files/file blocks
⇒ idea: store only once any given datum
⇒ used in: peer-to-peer file sharing, version control, etc.

Generic Lossless Compression

- well-known benefits (e.g., gzip, bzip2, etc.)
- unclear resource requirements

Techniques Not Considered

- differential compression: CPU- and memory-intensive, weakens data availability
- lossy compression: too specific (image, sound, etc.)
Chopping Data Into Small Blocks

Natural Solution: Fixed-Size Blocks

- simple and efficient
- similar data streams *might* yield common blocks

Finding More Similarities Using Content-Based Chopping

- see Udi Manber, *Finding Similar Files in a Large File System*, USENIX, 1994
- identifies **identical sub-blocks** among different data streams
- to be **coupled with single-instance storage**
- ⇒ improves storage efficiency? under **what circumstances**?
Providing a Suitable Meta-Data Format

Design Principle: Separation of Concerns

- separate data from meta-data
- separate stream meta-data from file meta-data

Indexing Individual Blocks

- avoid block name clashes
- block IDs must remain valid in time and space

Indexing Sequences of Blocks (illustrated)

- produce a vector of block IDs
- recursively chop it and index it
Providing Data Confidentiality, Integrity, and Authenticity

Enforcing Confidentiality

- encrypt both data & meta-data
- use energy-economic algorithms (e.g., symmetric encryption)

Allowing For Integrity Checks

- protect against both accidental and malicious modifications
- ⇒ store cryptographic hashes of (meta-)data blocks (e.g., SHA1, RIPEMD-160)
- ⇒ use hashes as a block naming scheme (content-based indexing)
- ⇒ eases implementation of single-instance storage

Allowing For Authenticity Checks

- cryptographically sign (part of) the meta-data
Enforcing Backup Atomicity

Comparison With Distributed and Mobile File Systems

- backup: only a single writer and reader
- thus, no consistency issues due to parallel accesses

Using Write-Once Semantics

- data is always appended, not modified
- previous versions are kept
- allows for atomic insertion of new data
- used in: peer-to-peer file sharing, version control
**Replication Using Erasure Codes**

**Erasure Codes at a Glance**

- $b$-block message $\rightarrow b \times S$ coded blocks
- $m$ blocks suffice to recover the message, $b < m < S \times b$
- $S \in \mathbb{R}$: stretch factor, overhead
- failures tolerated: $S \times b - m$
- $\implies$ More storage-efficient than simple replication

**Questions**

- Impact on data availability?
- Compared to simple replication?
Goals and Issues

Storage Mechanisms

Preliminary Evaluation of Storage Mechanisms

- Our Storage Layer Implementation: *libchop*
- Experimental Setup
- Algorithmic Combinations
- Storage Efficiency & Computational Cost Assessment
- Storage Efficiency & Computational Cost Assessment
**Our Storage Layer Implementation:** libchop

**Key Components**

- `chopper`, block & stream indexers, keyed block store
- provides several implementations of each component

**Strong Focus on Compression Techniques**

- **single-instance storage** (SHA-1-based block indexing)
- **content-based chopping** (Manber’s algorithm)
- **zlib compression filter** (similar to `gzip`)
Experimental Setup

Measurements

- storage efficiency
- computational cost (throughput)
- ... for different combinations of algorithms

File Sets

- a single mailbox file (low entropy)
- C program, several versions (low entropy, high redundancy)
- Ogg Vorbis files (high entropy, hardly compressable)
### Algorithmic Combinations

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A₁</td>
<td>no</td>
<td>—</td>
<td>—</td>
<td>yes</td>
<td>—</td>
</tr>
<tr>
<td>A₂</td>
<td>yes</td>
<td>—</td>
<td>—</td>
<td>yes</td>
<td>—</td>
</tr>
<tr>
<td>B₁</td>
<td>yes</td>
<td>Manber’s</td>
<td>1024 B</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>B₂</td>
<td>yes</td>
<td>Manber’s</td>
<td>1024 B</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>B₃</td>
<td>yes</td>
<td>fixed-size</td>
<td>1024 B</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>C</td>
<td>yes</td>
<td>fixed-size</td>
<td>1024 B</td>
<td>yes</td>
<td>no</td>
</tr>
</tbody>
</table>
## Storage Efficiency & Computational Cost Assessment

<table>
<thead>
<tr>
<th>Config.</th>
<th>Summary</th>
<th>Resulting Data Size</th>
<th>Throughput (MiB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>C files</td>
<td>Ogg</td>
</tr>
<tr>
<td>A₁</td>
<td>(without single instance)</td>
<td>26%</td>
<td>100%</td>
</tr>
<tr>
<td>A₂</td>
<td>(with single instance)</td>
<td>13%</td>
<td>100%</td>
</tr>
<tr>
<td>B₁</td>
<td>Manber</td>
<td>25%</td>
<td>102%</td>
</tr>
<tr>
<td>B₂</td>
<td>Manber + zipped blocks</td>
<td>11%</td>
<td>103%</td>
</tr>
<tr>
<td>B₃</td>
<td>fixed-size + zipped blocks</td>
<td>18%</td>
<td>103%</td>
</tr>
<tr>
<td>C</td>
<td>fixed-size + zipped input</td>
<td>13%</td>
<td>102%</td>
</tr>
</tbody>
</table>
Storage Efficiency & Computational Cost Assessment

Single-Instance Storage

- mostly beneficial in the **multiple version case** (50% improvement)
- computationally **inexpensive**

Content-Defined Blocks (Manber)

- mostly beneficial in the **multiple version case**
- computationally **costly**

Lossless Compression

- inefficient on high-entropy data (**Ogg** files)
- otherwise, **always beneficial** (block-level or whole-stream-level)
Conclusions

Implementation of a Flexible Prototype

- allows the combination of various storage techniques

Assessment of Compression Techniques

⇒ tradeoff between storage efficiency & computational cost
⇒ most suitable: lossless input compression + fixed-size chopping + single-instance storage

Six Essential Storage Requirements

- storage efficiency
- small data blocks
- backup atomicity
- error detection
- encryption
- backup redundancy
On-Going & Future Work

Improved Energetic Cost Assessment

- build on the **computational cost measurements** (execution time \( \approx \) energy)

Algorithmic Evaluation

- identify **tradeoffs in the replication/dissemination** processes (Markov chain analysis)
- develop algorithms to **dynamically adapt** to the environment (?)

Design & Implementation

- finalize the **overall architecture**
- integrate required technologies: **service discovery, authentication**, etc.
- interface with **trust management** mechanisms
Thank you!

Questions?

http://www.laas.fr/mosaic/

http://www.hidenets.aau.dk/