Wrapping up

Larry Rudolph

SMA 5508; MIT 6.833

We have only just …

- begun
- scratched the surface
- touched on the basics
- …
Some potentially relevant ideas

- Taken from my work in parallel processing
- Combining in network
- Amortized cost analysis
- Checkpointing
- Social Networks
- GuiDE
- You decide if they are relevant or useful

Atomicity

- First Goal: atomically update a variable
  \[ \text{atomic-add}( x, 17 ) \]
- Second Goal: update and use value
  - before any other process updates it
  \[ \text{res <-- fetch-add}( x, 17 ) \]

Initially, \( x \) is 100

Alice \( \leftarrow \text{FA}( x, 17 ) \)
print Alice

Bob \( \leftarrow \text{FA}( x, 3 ) \)
print Alice

\[ \gg 100 \quad (\text{or} \, 103) \]
\[ \gg 117 \quad (\text{or} \, 100) \]
Useful?

- critical sections
- concurrent list operations
  - don’t worry about wrap-around, just atomicity
  - what if 10,000 processors did it at same time?

```python
def pop:
    MyHead <-- FA( Head, 1 )
    return Q[MyHead]

def push(val):
    MyTail <-- FA( Tail, 1 )
    Q[MyTail] <-- val
```

Omega Network Routing

- Digit controlled:
  - start from any source
  - get to destination 0010
  - by following digits
  - “top, top, bottom, top” ports along the way
Collisions?

- if two packets enter each wanting same output port
- must buffer up one until the other leaves
- what if all want the same destination?
- all wanting access to “head” of linked list
- solution: combine within network

Combine within switch

\[ A = FA(X,17) \]
\[ B = FA(X, 3) \]
\[ 3 + 17 \]

\[ t = FA(X, 20) \]
\[ (t, A, B, 17) \]
\[ t = 100 \]
\[ (t, A, B, 17) \]
\[ A = 100 \]
\[ B = 117 \]
Is this relevant?

- NYU Ultracomputer --> IBM RP3 --> IBM SP series
- Do one thing right?
  - no, do everything right
  - have a unifying “gem” or “nugget” to galvanize team
- Relevant to pervasive computing?
  - Perhaps -- depends on application

How to decide?

- Nobody knows the future, but need to decide anyway
- Skiing analogy:
  - $50 to rent; $400 to buy
  - should you rent or buy?
- Alice and Bob share a computer
  - $5 in bus fare for Bob to get to Alice; $500 to buy
  - should Bob use Alice’s computer or buy his own?
Amortized Cost Analysis

- When you pay enough renting, then buy
- Never pay more than a factor of 2 no matter what
- Can do better only if know something about future
- Alice & Bob can buy and throw away based on usage
- Useful to decide to cache locally or access remotely
- (of course other considerations: snoopy caching)

Snapshots, checkpoints, rollback, and restart

Larry Rudolph
MIT CSAIL
January 2005
Overview

Computers fail
When they fail, work gets lost
Should save work often to reduce the pain
But I back up my disk only when I hear that someone else’s disk crashed
Does this make sense?

Checkpoints

This talk is concerned about running programs crashing (not disks)
We use the term “checkpoint” to be
a process that stores sufficient program state to nonvolatile storage (disk) so that the program can be restarted from that point
Do Checkpoints matter?

- For short computations, just rerun -- NO
- For long running computations -- YES
  - An OS runs long but it doesn’t matter
- For parallel programs, it is harder (interesting)
  - Stop all processes and all communication
  - Write all relevant state to disk

OS does all the important stuff

- Checkpointing incurs overhead
- stops computation; uses network & disk bw
- System initiated (helps users, system pays)
- airlines would rather you bought another ticket if flight was cancelled
- Application initiated (looking out for #1)
- Knows best places, e.g. outer loop
Checkpointing Overhead

Assume periodic checkpointing
- done every I work period
- the overhead of checkpointing is C
When failure happens, redo work since ckpt

Don’t do checkpoints if C > I
Surprisingly, our study show that this improves performance of existing systems!
Key Idea: Just say no (if you don’t want to do it)

- Application initiates checkpoints (at best place)
- It asks the system to actually perform it
- The system can decide to not do it
- The system knows more about the overall system state than the application

---

Key Idea: Just say no (if you don’t want to)

- = Worst-case failure
- k = Working on block k
- = Checkpointing

Diagram:
- Ideal
- Perform Checkpoint
- Skip Checkpoint

Time
Risk-based decision

Recall, skip checkpoint if \( I < C \) (not worth it)

Really want to say:

Cost of doing ckpt vs cost of not doing ckpt

Let \( p \) be probability of a failure

skip checkpoint if \( pI < C \)

Failure prediction

Examination of logs, can predict failures

based on temp, power spikes, retries, alarms

have already shown success rate of 70%

System has good idea of “\( p \)”

System has good idea of cost of failure
Checkpoint in message-passing systems

Each processor or node does a **checkpoint** of its local state.

The system takes a **snapshot**, checkpoints plus messages not yet part of state.
Main Memory

Cache

Evict Entries upon Context Switch
**Cache Partitioning -- “Column Caching”**

- Common case: no overhead
- On miss (rare event)
  Usable Columns Info in TLB
  No extra latency

**Search All Columns**

<table>
<thead>
<tr>
<th>Ta</th>
<th>Dat</th>
</tr>
</thead>
<tbody>
<tr>
<td>g</td>
<td>a</td>
</tr>
</tbody>
</table>

**Replace only in selected columns**

<table>
<thead>
<tr>
<th>TL</th>
<th>Column Spec</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>1 0 0 0</td>
</tr>
<tr>
<td></td>
<td>0 0 1 0</td>
</tr>
<tr>
<td></td>
<td>0 0 0 1</td>
</tr>
</tbody>
</table>

**Cache RePartitioning -- Easy if always search all cols**

- Common case: no overhead
- On miss (rare event)
  Usable Columns Info in TLB
  No extra latency

**Search All Columns**

<table>
<thead>
<tr>
<th>Ta</th>
<th>Dat</th>
</tr>
</thead>
<tbody>
<tr>
<td>g</td>
<td>a</td>
</tr>
</tbody>
</table>

**Replace only in selected columns**

<table>
<thead>
<tr>
<th>TL</th>
<th>Column Spec</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>1 0 0 0</td>
</tr>
<tr>
<td></td>
<td>0 1 1 0</td>
</tr>
<tr>
<td></td>
<td>0 0 0 1</td>
</tr>
<tr>
<td></td>
<td>0 0 0 1</td>
</tr>
</tbody>
</table>
Streaming Data (Conventional Cache)

Processor explicitly communicates with external interface

Buffer required in case of slow processor

Data pushed into DRAM by Interface

Data pulled from DRAM by Processor
Curious Caching

Insert data into cache as dirty

Store 1,B
Store 1,A[2]
Store 1,X
Store 1,A[1]

Insert data into cache as clean

DRAM

Cache full, so store as usual

Column + Curious Caching

On Chip Memory (SRAM)

One column of addresses exclusively allocated to one column of cache

Column mark curious&dirty

Curious Address Ranges

5000:5999 (clean)
3010:4123 (dirty)

Curious Address Ranges

5000:5999 (clean)
3010:3011 (dirty)

Column

On Chip Memory (SRAM)

One column of addresses exclusively allocated to one column of cache

Column mark curious&dirty

TL

Spec

- - - - - -
0 0 0 0
- - - - - -
0 1 1 0
- - - - - -
0 0 0 1
Social Networks

How does information flow in an organization
- Email logs (edge from sender to receiver)
- Forums (edge from responder to requester)
- Telephone logs
- Anonymize the logs
- Patterns in static or dynamic graph
  - We can distinguish
    - production mode (tree-like exchanges)
    - innovative mode (power-law static; oscillation in dynamic graph)
Real communications cluster by category and by geography

Content Summary
- The x-axis represents time
- Categories generated by the SOM are laid vertically
- Each green bar represents one message
- The thickness of each sub-topic indicates its activity on particular day
- The length in the x-dimension of each sub-topic = time duration of that sub-topic

Visual Effects:
- Thickness = how active a subtopic is
- Length in x-dimension = the time duration of a sub-topic
**E-Mail Analysis**

Zoomable View of Inferred Social Network
Currently visualizing search results for the terms “California” (yellow), “FERC” (orange), or both (red).

Color-coded categorization of e-mail content
Navigable timeline view of e-mail traffic
Interactive e-mail analysis environment

Data: Enron E-Mail Corpus, over 400,000 emails

---

**TeCFlow**

Intermittent
Growing Interest
More Stable
Face-to-face Information Flow

- Capture physical interactions
- Infrastructure & Privacy Challenges
  - Bluetooth beacons throughout environment
  - Cell phone knows its location from beacon
  - Easy to deploy ($30+AC outlet per beacon)

Privacy Concerns

- People do not like to be tracked
- Cannot tell if Alice spends times in Bob’s office
- Phone records location every k minutes
- Universal hash of (time, location)
  - Appears like a sequence of random numbers
  - Distributed according to “n balls in m buckets”
- Server periodically gets sequence from phone
  - People at same place / same time
- Can infer many meeting patterns from distribution
- Cannot infer actions of any individual!
Physical Location Tracking

Probability pattern of interactions, no individual information can be determined

Different hash functions (under phone control) can leak some individual location information

Can be used to verify other data sets inferences

E.g. hours worked from email logs

Infer lots from how patterns change over time
**A SmartMobile GuiDE**

**Goals:**
- Provide services, functions, apps, data on phones
- Maintain Structured Data Representation to
  1. Search and Query
  2. Manipulate
- Universal (works on many phone models)

**Phone GuiDE:**
GuiDE is the underlying data structure
- GUI - Graphical User Interface
- Decode - window objects and actions
- Enrich - add extra information to nodes

**Key Insight:**
Although there are published API's for accessing phone functions and information, and for accessing and setting configurations, the GUI is what always works.

- Provides enough context for user to understand
- Easy to identify three types of text (and more)
- Maintain a graph structure (mostly tree-like)