CS 347:
Parallel and Distributed
Data Management

Notes X: S4

Hector Garcia-Molina

Material based on:

2010 IEEE International Conference on Data Mining Workshops

S4: Distributed Stream Computing Platform

Leonardo Neumeyer
Yahoo! Labs
Santa Clara, CA
neumeyer@yahoo-inc.com

Bruce Robbins
Yahoo! Labs
Santa Clara, CA
robbins@yahoo-inc.com

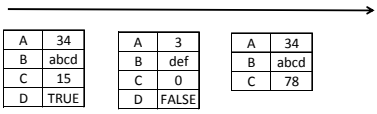
Anish Nair
Yahoo! Labs
Santa Clara, CA
anishn@yahoo-inc.com

Anand Kesari
Yahoo! Labs
Santa Clara, CA
anands@yahoo-inc.com

S4

- Platform for processing unbounded data <u>streams</u>
  - general purpose
  - distributed
  - scalable
  - partially fault tolerant
    (whatever this means!)

Data Stream

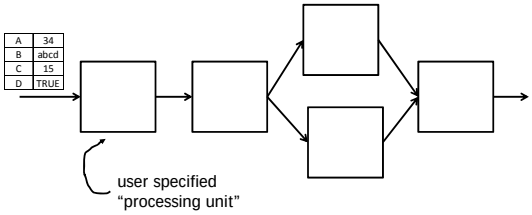


Terminology: event (data record), key, attribute

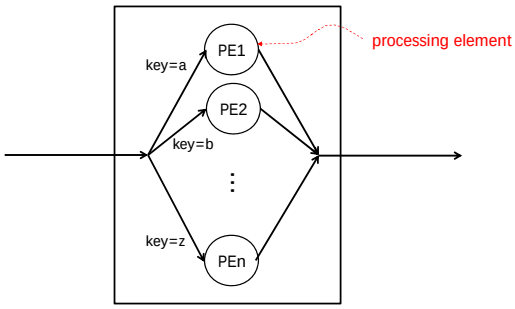
Question: Can an event have duplicate attributes for same key?
(I think so...)

Stream unbounded, generated by say user queries,
purchase transactions, phone calls, sensor readings,...

S4 Processing Workflow

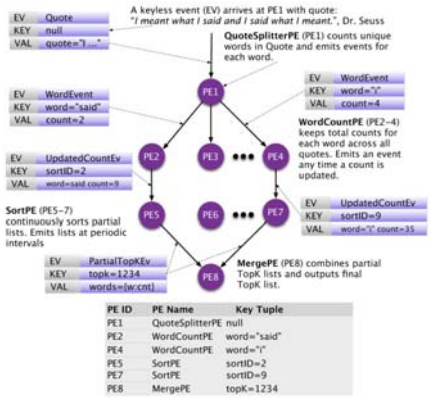


Inside a Processing Unit

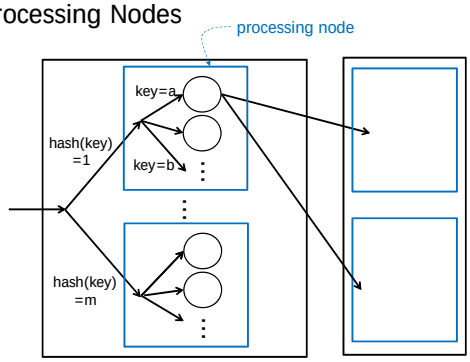


Example:

- Stream of English quotes
- Produce a sorted list of the top K most frequent words



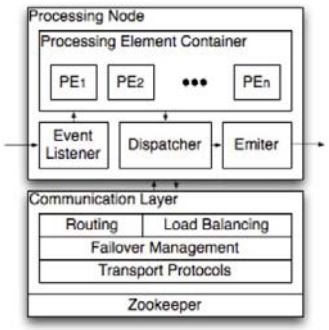
Processing Nodes



Dynamic Creation of PEs

- As a processing node sees new key attributes, it dynamically creates new PEs to handle them
- Think of PEs as threads

Another View of Processing Node

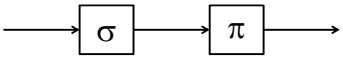


Failures

- Communication layer detects node failures and provides failover to standby nodes
- What happens events in transit during failure? (My guess: events are lost!)

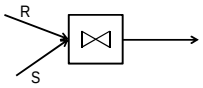
How do we do DB operations on top of S4?

- Selects & projects easy!



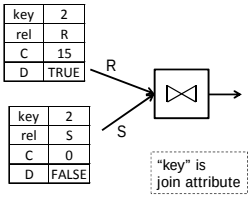
- What about joins?

What is a Stream Join?



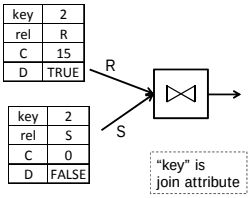
- For true join, would need to store <u>all</u> inputs forever! Not practical…
- Instead, define window join:
  - at time t new R tuple arrives
  - it only joins with previous w S tuples

One Idea for Window Join



<u>code for PE:</u>
for each event e:
  if e.rel=R
    [store in Rset(last w)
      for s in Sset:
        output join(e,s)]
  else …

One Idea for Window Join

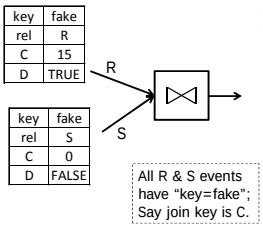


<u>code for PE:</u>
for each event e:
  if e.rel=R
    [store in Rset(last w)
      for s in Sset:
        output join(e,s)]
  else …

Is this right???
(enforcing window on a per-key value basis)

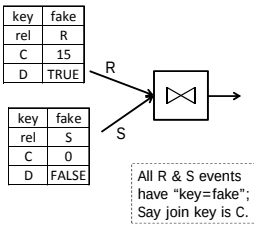
Maybe add sequence numbers to events to enforce correct window?

Another Idea for Window Join



<u>code for PE:</u>
for each event e:
  if e.rel=R
    [store in Rset(last w)
      for s in Sset:
        if e.C=s.C then
          output join(e,s)]
  else if e.rel=S …

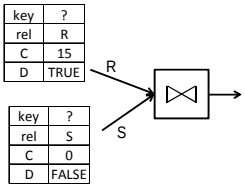
Another Idea for Window Join



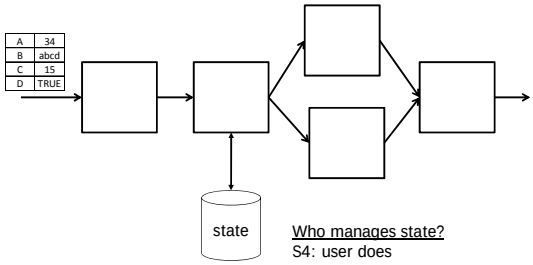
<u>code for PE:</u>
for each event e:
  if e.rel=R
    [store in Rset(last w)
      for s in Sset:
        if e.C=s.C then
          output join(e,s)]
  else if e.rel=S …

Entire join done in one PE; no parallelism

Do You Have a Better Idea for Window Join?



Final Comment: Managing state of PE



Who manages state?
S4: user does
Mupet: System does
Is state persistent?

CS 347: Parallel and Distributed Data Management

Notes X: Hyracks

Hector Garcia-Molina

Hyracks

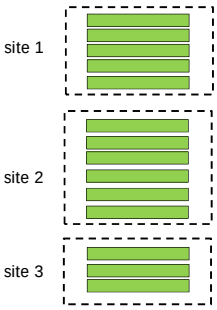
- Generalization of map-reduce
- Infrastructure for "big data" processing
- Material here based on:

Hyracks: A Flexible and Extensible Foundation for Data-Intensive Computing

Vinayak Borkar, Michael Carey, Raman Grover, Nicola Onose, Rares Vernica
Computer Science Department, University of California, Irvine
Irvine, CA 92697
vborkar@ics.uci.edu

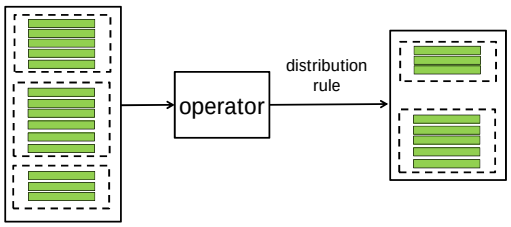
Appeared in ICDE 2011

A Hyracks data object:

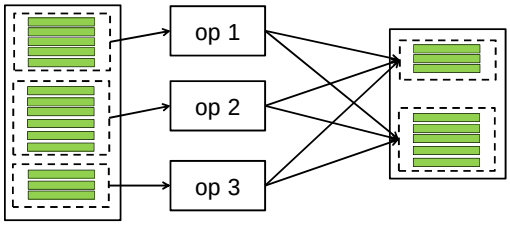


- Records partitioned across N sites
- Simple record schema is available (more than just key-value)

Operations



Operations (parallel execution)



Example: Hyracks Specification

select C_MKTSEGMENT, count (O_ORDERKEY)
from CUSTOMER **join** ORDERS **on** C_CUSTKEY = O_CUSTKEY
group by C_MKTSEGMENT

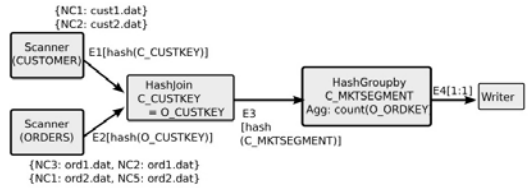


Fig. 1: Example Hyracks job specification

Example: Hyracks Specification

select C_MKTSEGMENT, count (O_ORDERKEY)
from CUSTOMER **join** ORDERS **on** C_CUSTKEY = O_CUSTKEY
group by C_MKTSEGMENT

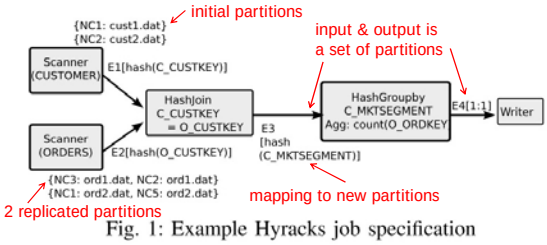


Fig. 1: Example Hyracks job specification

Notes

- Job specification can be done manually or automatically

Example: Activity Node Graph

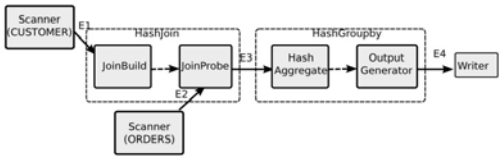


Fig. 2: Example Hyracks Activity Node graph

Example: Activity Node Graph

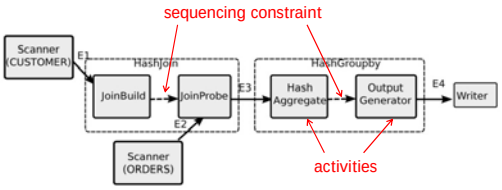


Fig. 2: Example Hyracks Activity Node graph

Example: Parallel Instantiation

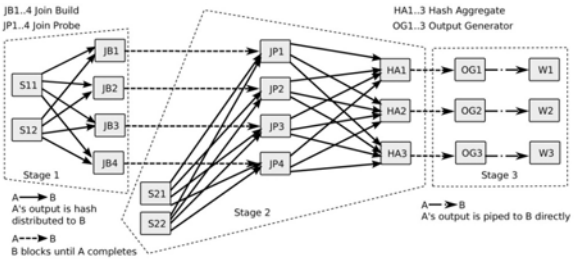


Fig. 3: Parallel instantiation of the example

Example: Parallel Instantiation

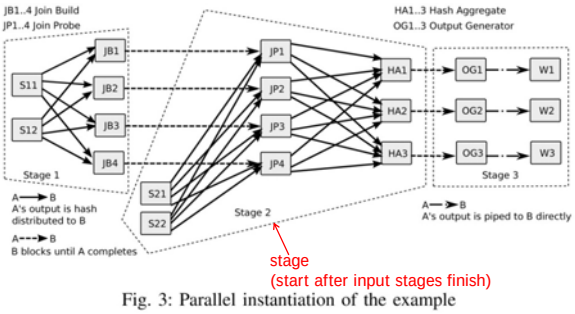


Fig. 3: Parallel instantiation of the example

System Architecture

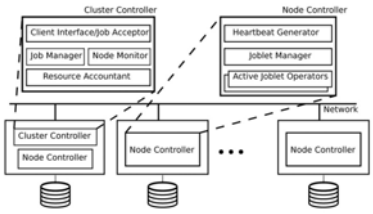


Fig. 4: Hyracks system architecture

Map-Reduce on Hyracks

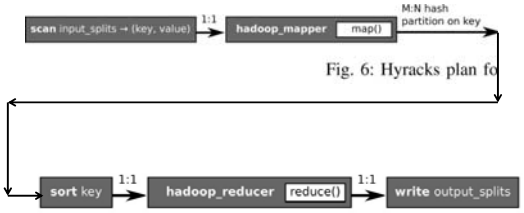


Fig. 6: Hyracks plan fo

r Hadoop jobs

Library of Operators:

- File reader/writers
- Mappers
- Sorters
- Joiners (various types0
- Aggregators

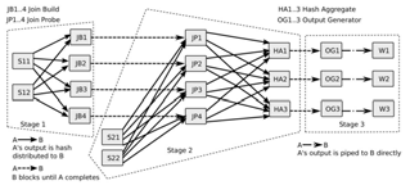
- Can add more

Library of Connectors:

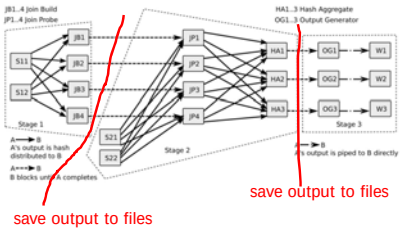
- N:M hash partitioner
- N:M hash-partitioning merger (input sorted)
- N:M rage partitioner (with partition vector)
- N:M replicator
- 1:1

- Can add more!

Hyracks Fault Tolerance: Work in progress?

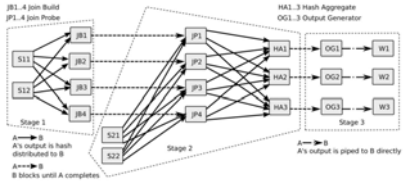


Hyracks Fault Tolerance: Work in progress?



The Hadoop/MR approach: save partial results to persistent storage; after failure, redo all work to reconstruct missing data

Hyracks Fault Tolerance: Work in progress?



Can we do better?
Maybe: each process retains previous results until no longer needed?

Pi output: ~~r1, r2, r3,~~ r4, r5, r6, r7, r8

have "made way" to final result

current output

CS 347: Parallel and Distributed Data Management

Notes X: Pregel

Hector Garcia-Molina

Material based on:

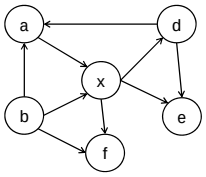
Pregel: A System for Large-Scale Graph Processing

Grzegorz Malewicz, Matthew H. Austern, Aart J. C. Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski
Google, Inc.
{malewicz,austern,ajcbik,dehnert,ilan,naty,gczajj}@google.com

- In SIGMOD 2010
- Note there is an open-source version of Pregel called GIRAPH

Pregel

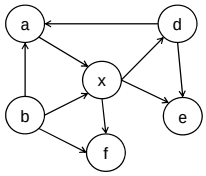
- A computational model/infrastructure for processing large graphs
- Prototypical example: Page Rank



PR[i+1,x] = f(PR[i,a]/na, PR[i,b]/nb)

Pregel

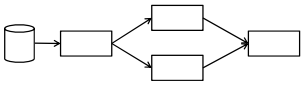
PR[i+1,x] =
f(PR[i,a]/na, PR[i,b]/nb)



- Synchronous computation in iterations
- In one iteration, each node:
  - gets messages from neighbors
  - computes
  - sends data to neighbors

Pregel vs Map-Reduce/S4/Hyracks/…

- In Map-Reduce, S4, Hyracks,… workflow separate from data



- In Pregel, data (graph) drives data flow

Pregel Motivation

- Many applications require graph processing
- Map-Reduce and other workflow systems not a good fit for graph processing
- Need to run graph algorithms on many procesors

Example of Graph Computation

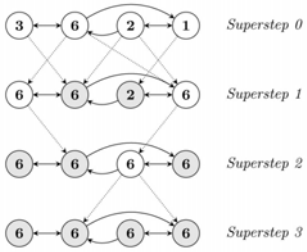


Figure 2: Maximum Value Example. Dotted lines are messages. Shaded vertices have voted to halt.

Termination

- After each iteration, each vertex votes to halt or not
- If <u>all</u> vertexes vote to halt, computation terminates

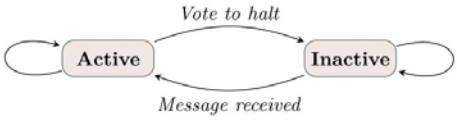


Figure 1: Vertex State Machine

Vertex Compute (simplified)

- Available data for iteration i:
  - InputMessages: { [from, value] }
  - OutputMessages: { [to, value] }
  - OutEdges: { [to, value] }
  - MyState: value
- OutEdges and MyState are remembered for next iteration

Max Computation

- change := false
- for [f,w] in InputMessages do
  if w > MyState.value then
  [MyState.value := w
  change := true]
- if (superstep = 1) OR change then
  for [t, w] in OutEdges do
  add [t, MyState.value] to OutputMessages
  else vote to halt

Page Rank Example

```
class PageRankVertex
    : public Vertex<double, void, double> {
 public:
  virtual void Compute(MessageIterator* msgs) {
    if (superstep() >= 1) {
      double sum = 0;
      for (; !msgs->Done(); msgs->Next())
        sum += msgs->Value();
      *MutableValue() =
          0.15 / NumVertices() + 0.85 * sum;
    }

    if (superstep() < 30) {
      const int64 n = GetOutEdgeIterator().size();
      SendMessageToAllNeighbors(GetValue() / n);
    } else {
      VoteToHalt();
    }
  }
};
```


Page Rank Example

```
class PageRankVertex
    : public Vertex<double, void, double> {
 public:
  virtual void Compute(MessageIterator* msgs) {
    if (superstep() >= 1) {
      double sum = 0;
      for (; !msgs->Done(); msgs->Next())
        sum += msgs->Value();
      *MutableValue() =
          0.15 / NumVertices() + 0.85 * sum;
    }

    if (superstep() < 30) {
      const int64 n = GetOutEdgeIterator().size();
      SendMessageToAllNeighbors(GetValue() / n);
    } else {
      VoteToHalt();
    }
  }
};
```

iteration count

iterate thru InputMessages

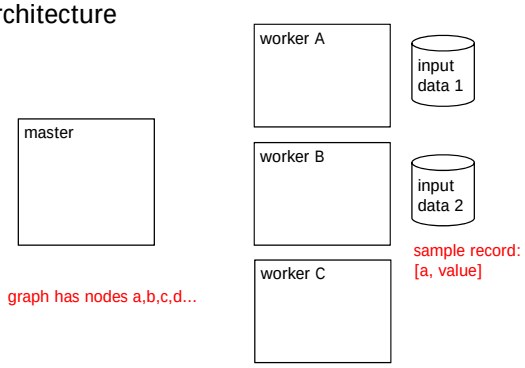
MyState.value

shorthand: send same msg to all

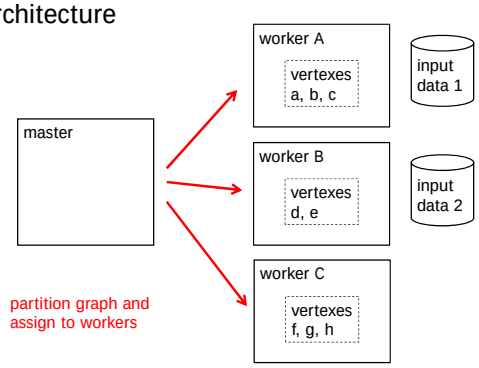
Single-Source Shortest Paths

```
class ShortestPathVertex
    : public Vertex<int, int, int> {
  void Compute(MessageIterator* msgs) {
    int mindist = IsSource(vertex_id()) ? 0 : INF;
    for (; !msgs->Done(); msgs->Next())
      mindist = min(mindist, msgs->Value());
    if (mindist < GetValue()) {
      *MutableValue() = mindist;
      OutEdgeIterator iter = GetOutEdgeIterator();
      for (; !iter.Done(); iter.Next())
        SendMessageTo(iter.Target(),
                      mindist + iter.GetValue());
    }
    VoteToHalt();
  }
};
```

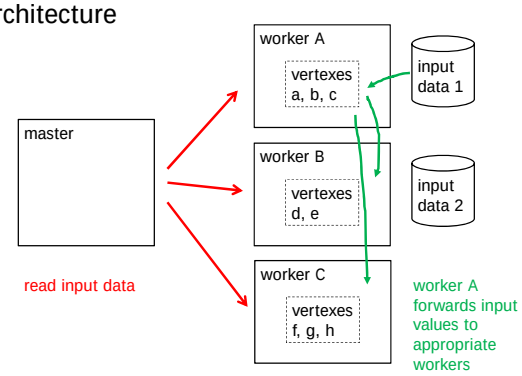

Architecture



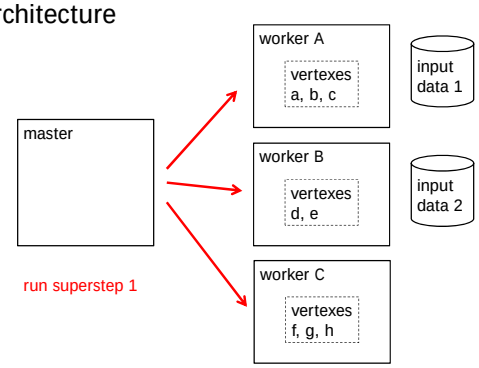
Architecture



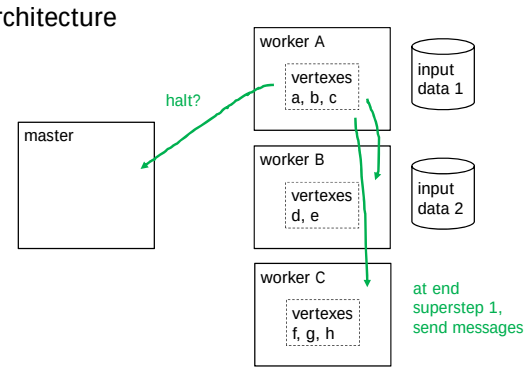
Architecture



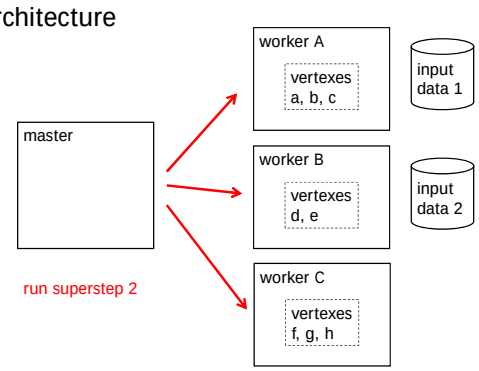
Architecture



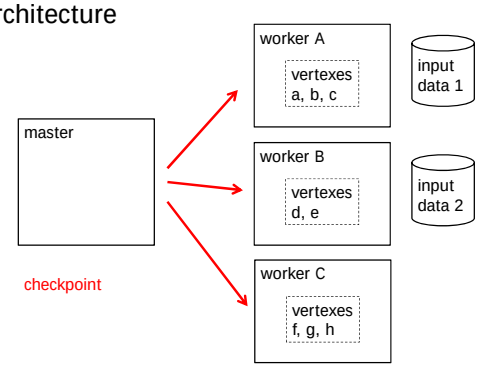
Architecture



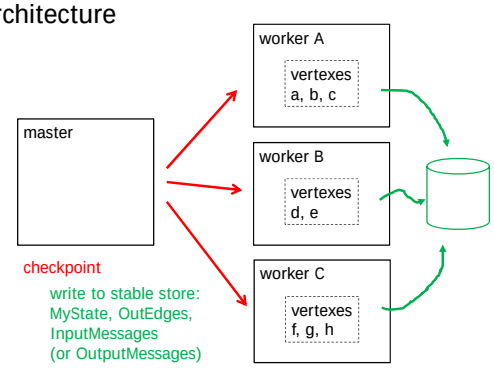
Architecture



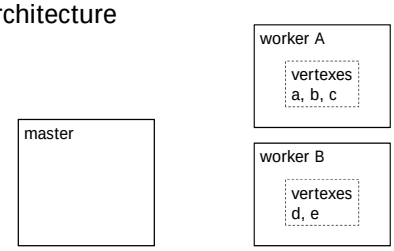
Architecture



Architecture

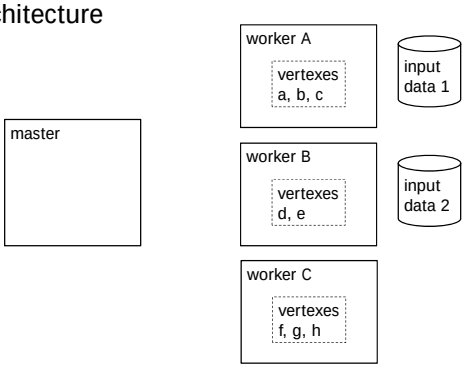


Architecture



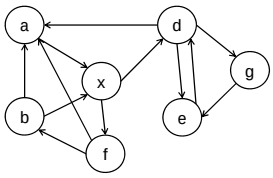
if worker dies,
find replacement &
restart from
latest checkpoint

Architecture



Interesting Challenge

- How best to partition graph for efficiency?



CS 347:
Parallel and Distributed
Data Management

Notes X:
BigTable, HBASE, Cassandra

Hector Garcia-Molina

Sources

- HBASE: The Definitive Guide, Lars George, O'Reilly Publishers, 2011.
- Cassandra: The Definitive Guide, Eben Hewitt, O'Reilly Publishers, 2011.
- BigTable: A Distributed Storage System for Structured Data, F. Chang et al, ACM Transactions on Computer Systems, Vol. 26, No. 2, June 2008.

Lots of Buzz Words!

- "Apache Cassandra is an open-source, distributed, decentralized, elastically scalable, highly available, fault-tolerant, tunably consistent, column-oriented database that bases its distribution design on Amazon's dynamo and its data model on Google's Big Table."
- Clearly, it is buzz-word compliant!!

Basic Idea: Key-Value Store

Table T:

key	value
k1	v1
k2	v2
k3	v3
k4	v4

Basic Idea: Key-Value Store

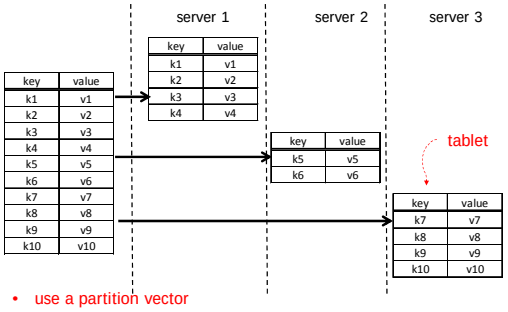
Table T:

key	value
k1	v1
k2	v2
k3	v3
k4	v4

keys are sorted

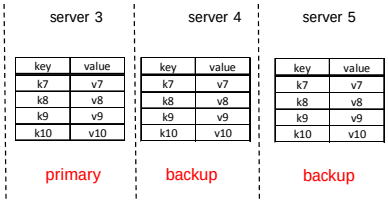
- API:
  - lookup(key) → value
  - lookup(key range) → values
  - getNext → value
  - insert(key, value)
  - delete(key)
- Each row has timestemp
- Single row actions atomic
  (but not persistent in some systems?)
- No multi-key transactions
- No query language!

Fragmentation (Sharding)



- use a partition vector
- "auto-sharding": vector selected automatically

Tablet Replication



- Cassandra:
  Replication Factor (# copies)
  R/W Rule: One, Quorum, All
  Policy (e.g., Rack Unaware, Rack Aware, ...)
  Read all copies (return fastest reply, do repairs if necessary)
- HBase: Does not manage replication, relies on HDFS

Need a "directory"

- Table Name: Key → Server that stores key
  → Backup servers
- Can be implemented as a special table.

Tablet Internals

memory

key	value
k3	v3
k8	v8
k9	delete
k15	v15

disk

key	value
k2	v2
k6	v6
k9	v9
k12	v12

key	value
k4	v4
k5	delete
k10	v10
k20	v20
k22	v22

Design Philosophy (?): Primary scenario is where all data is in memory.
Disk storage added as an afterthought

Tablet Internals

tombstone

key	value
k3	v3
k8	v8
k9	delete
k15	v15

memory

flush periodically

disk

key	value
k2	v2
k6	v6
k9	v9
k12	v12

key	value
k4	v4
k5	delete
k10	v10
k20	v20
k22	v22

- tablet is merge of all segments (files)
- disk segments imutable
- writes efficient; reads only efficient when all data in memory
- periodically reorganize into single segment

Column Family

K	A	B	C	D	E
k1	a1	b1	c1	d1	e1
k2	a2	null	c2	d2	e2
k3	null	null	null	d3	e3
k4	a4	b4	c4	e4	e4
k5	a5	b5	null	null	null

Column Family

K	A	B	C	D	E
k1	a1	b1	c1	d1	e1
k2	a2	null	c2	d2	e2
k3	null	null	null	d3	e3
k4	a4	b4	c4	e4	e4
k5	a5	b5	null	null	null

- for storage, treat each row as a single "super value"
- API provides access to sub-values
  (use family:qualifier to refer to sub-values
  e.g., price:euros, price:dollars)
- Cassandra allows "super-column":
  two level nesting of columns
  (e.g., Column A can have sub-columns X & Y)

Vertical Partitions

K	A	B	C	D	E
k1	a1	b1	c1	d1	e1
k2	a2	null	c2	d2	e2
k3	null	null	null	d3	e3
k4	a4	b4	c4	e4	e4
k5	a5	b5	null	null	null

can be manually implemented as

K	A
k1	a1
k2	a2
k4	a4
k5	a5

K	B
k1	b1
k4	b4
k5	b5

K	C
k1	c1
k2	c2
k4	c4

K	D	E
k1	d1	e1
k2	d2	e2
k3	d3	e3
k4	e4	e4

Vertical Partitions

K	A	B	C	D	E
k1	a1	b1	c1	d1	e1
k2	a2	null	c2	d2	e2
k3	null	null	null	d3	e3
k4	a4	b4	c4	e4	e4
k5	a5	b5	null	null	null

column family

K	A
k1	a1
k2	a2
k4	a4
k5	a5

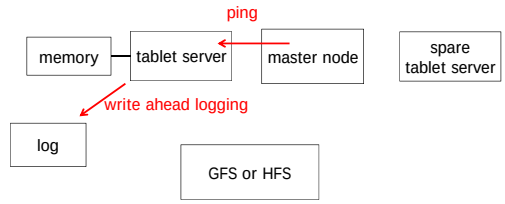
K	B
k1	b1
k4	b4
k5	b5

K	C
k1	c1
k2	c2
k4	c4

K	D	E
k1	d1	e1
k2	d2	e2
k3	d3	e3
k4	e4	e4

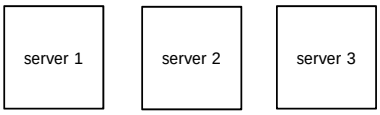
- good for sparse data;
- good for column scans
- not so good for tuple reads
- are atomic updates to row still supported?
- API supports actions on full table; mapped to actions on column tables
- API supports column "project"
- To decide on vertical partition, need to know access patterns

Failure Recovery (BigTable, HBase)



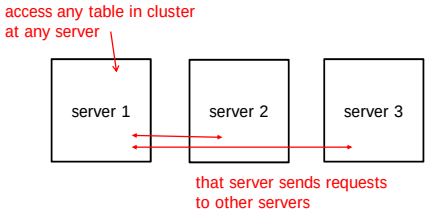
Failure recovery (Cassandra)

- No master node, all nodes in “cluster” equal



Failure recovery (Cassandra)

- No master node, all nodes in “cluster” equal



CS 347: Parallel and Distributed Data Management

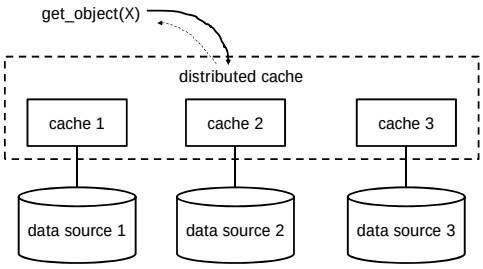
Notes X: MemCacheD

Hector Garcia-Molina

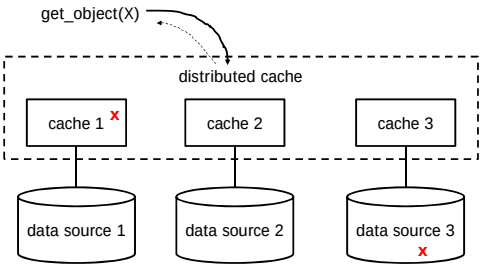
MemCacheD

- General-purpose distributed memory caching system
- Open source

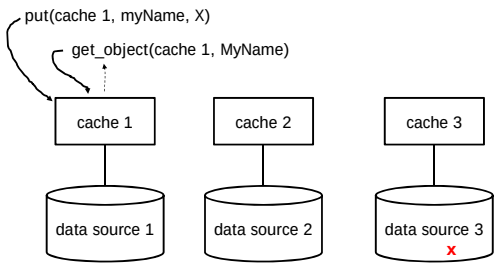
What MemCacheD Should Be (but ain't)



What MemCacheD Should Be (but ain't)

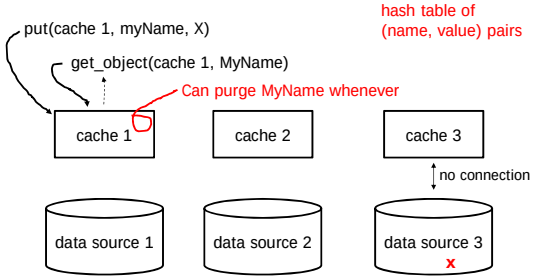


What MemCacheD Is (as far as I can tell)



What MemCacheD Is

each cache is hash table of (name, value) pairs



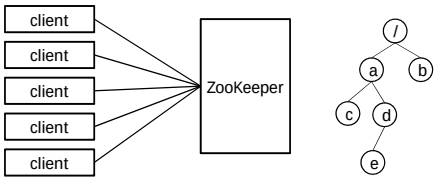
CS 347: Parallel and Distributed Data Management

Notes X: ZooKeeper

Hector Garcia-Molina

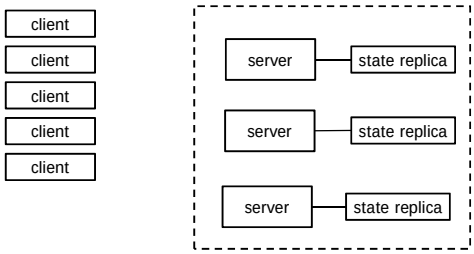
ZooKeeper

- Coordination service for distributed processes
- Provides clients with high throughput, high availability, memory only file system

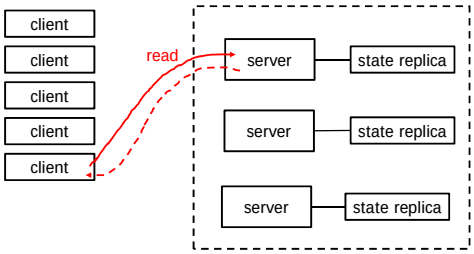


znode /a/d/e (has state)

ZooKeeper Servers

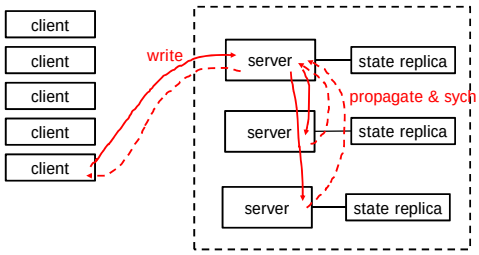


ZooKeeper Servers



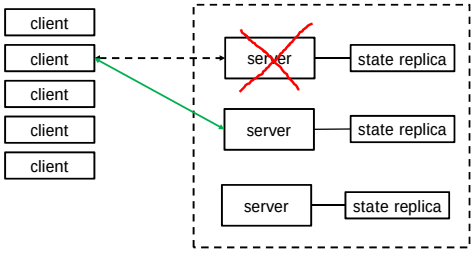
ZooKeeper Servers

writes totally ordered:
used Zab algorithm



Failures

if your server dies,
just connect to a different one!



ZooKeeper Notes

- Differences with file system:
  - all nodes can store data
  - storage size limited
- API: insert node, read node, read children, delete node, ...
- Can set triggers on nodes
- Clients and servers must know all servers
- ZooKeeper works as long as a majority of servers are available
- Writes totally ordered; read ordered w.r.t. writes

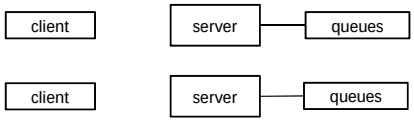
CS 347: Parallel and Distributed Data Management

Notes X: Kestrel

Hector Garcia-Molina

Kestrel

- Kestrel server handles a set of reliable, ordered message queues
- A server does not communicate with other servers (advertised as good for scalability!)



Kestrel

- Kestrel server handles a set of reliable, ordered message queues
- A server does not communicate with other servers (advertised as good for scalability!)