The Gamma Database Machine Project

David DeWitt, Shahram Ghandeharizadeh, Donovan Schcheider,
Allan Bricker, Hui-i Hsiao, and Rick Rasmussen

Slides adopted from those of Deepak Bastakoty,
and Ghandeharizadeh and DeWitt, Jianhao Cao

Presenter: Tanya Prasad
Discussion Leader: Jonas Tai
UBC CPSC 504 – 2023.03.06

Motivation

- Why parallel databases?
  - Obtain faster response time
  - Increase query throughput
  - Improve robustness to failure
  - Reduce processor workload
  - Enable scalability

Motivation

- DIRECT
  - Early parallel database project
  - Shared memory
  - Centralized control of parallel algorithms

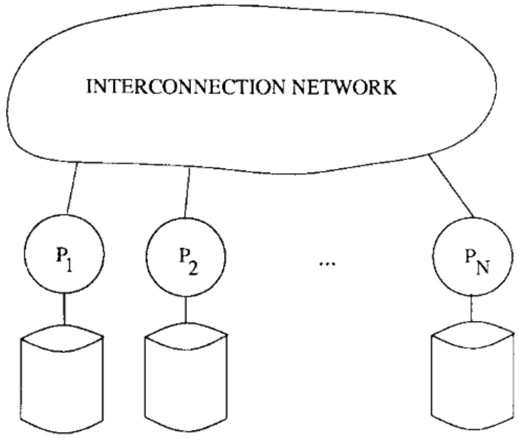
Motivation

- DIRECT
  - Early parallel database project
  - ➡ Shared memory
  - ➡ Centralized control of parallel algorithms

Impossible to scale the architecture to hundreds of processors!

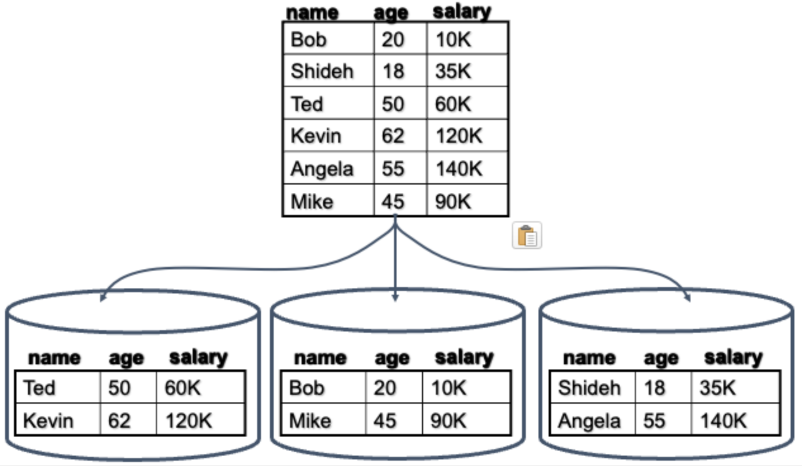
Motivation

- Share-nothing
  - Each processor has it own memory or disk(s)
- Hash-based parallel algorithms
  - No need for centralized control



Motivation

- Horizontal partitioning (declustering)
  - Tuples of a relation distributed over multiple disks.
  - Round robin; hashed; range partitioned



name	age	salary
Bob	20	10K
Shideh	18	35K
Ted	50	60K
Kevin	62	120K
Angela	55	140K
Mike	45	90K

name	age	salary
Ted	50	60K
Kevin	62	120K

name	age	salary
Bob	20	10K
Mike	45	90K

name	age	salary
Shideh	18	35K
Angela	55	140K

Hardware Architecture

❑ GAMMA 1.0

- 17 VAX 11/750 processors, each with 2 MB memory
- Another VAX as the host machine
- An 80 Mb/s token ring to connect processors
- 8 processors attached with 333 MB disk drivers

❑ Problems

- The token ring network packet size is too small (2K bytes)
- The bandwidth mismatch between the token ring and the Unibus on the 11/750
- Insufficient memory for each processor

Hardware Architecture

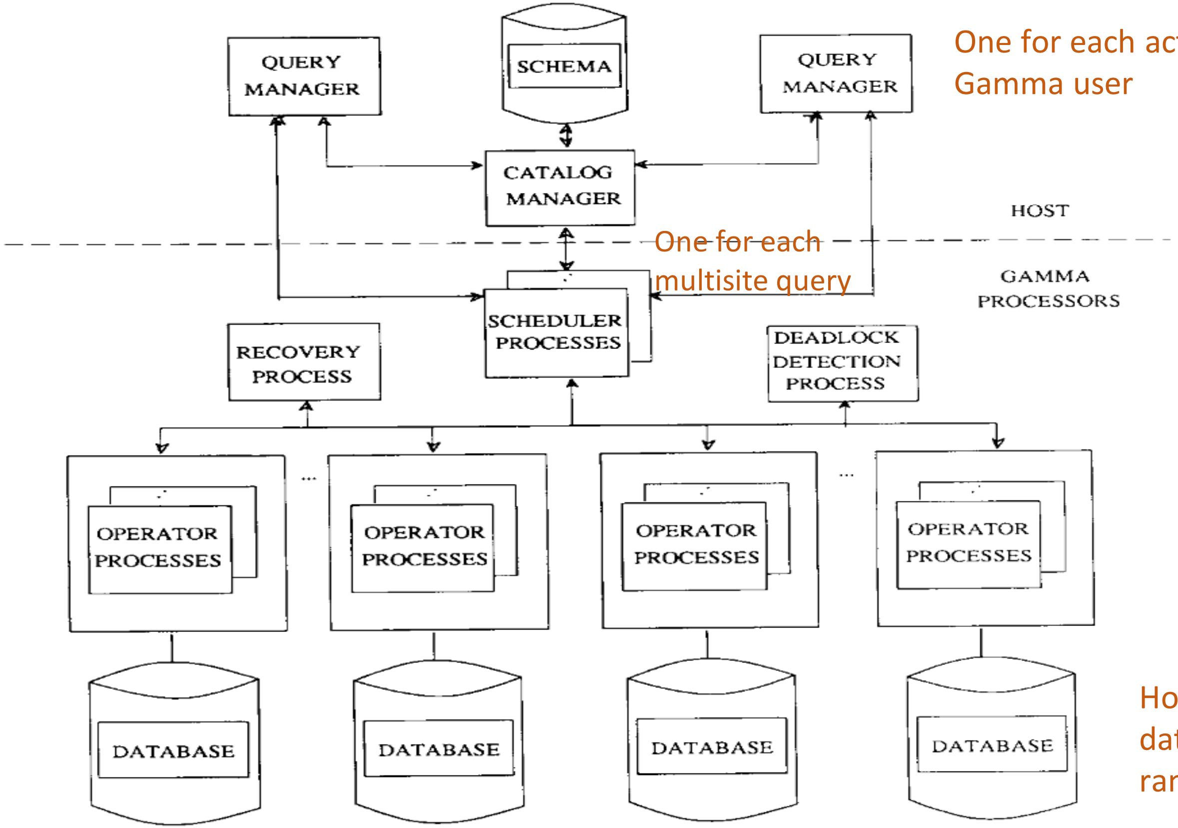
❑ GAMMA 2.0

- 32 processor iPSC/2 hypercube from Intel
- 386 CPU, 8 MB memory
- 330 MB MAXTOR 4380 disk drive with a 45 KB RAM buffer
- Custom VLSI routing modules for network communication
- NOSE (Gamma’s OS) run as a thread package inside a process

Discussion 1 (Groups of 3, at least 1 Systems)

- As some of you pointed out in their reviews, the authors spend a lot of time talking about hardware
  - Issues in Gamma Version 1.0 such as insufficient memory
  - Problems with the disk controller in Gamma Version 2.0
  - Conversion problems because of different addressing schemes
- What do you think was the motivation to include this long section about the hardware and the problems they faced?
- Do you think the experiences they made with the chosen hardware strengthen, weaken or do not impact the paper?

Software Architecture



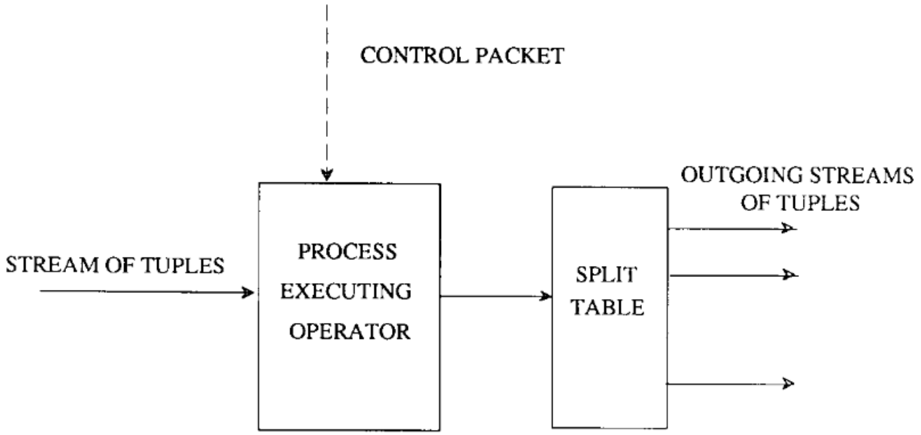
One for each active Gamma user

One for each multisite query

Horizontally partitioned data: round robin; hashed; range partitioned

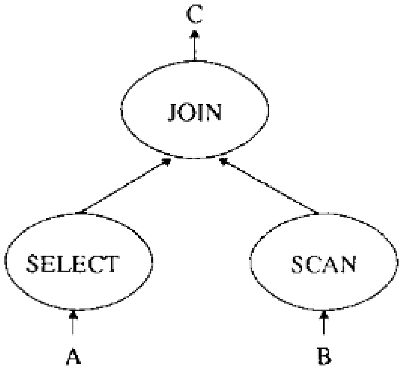
Software Architecture

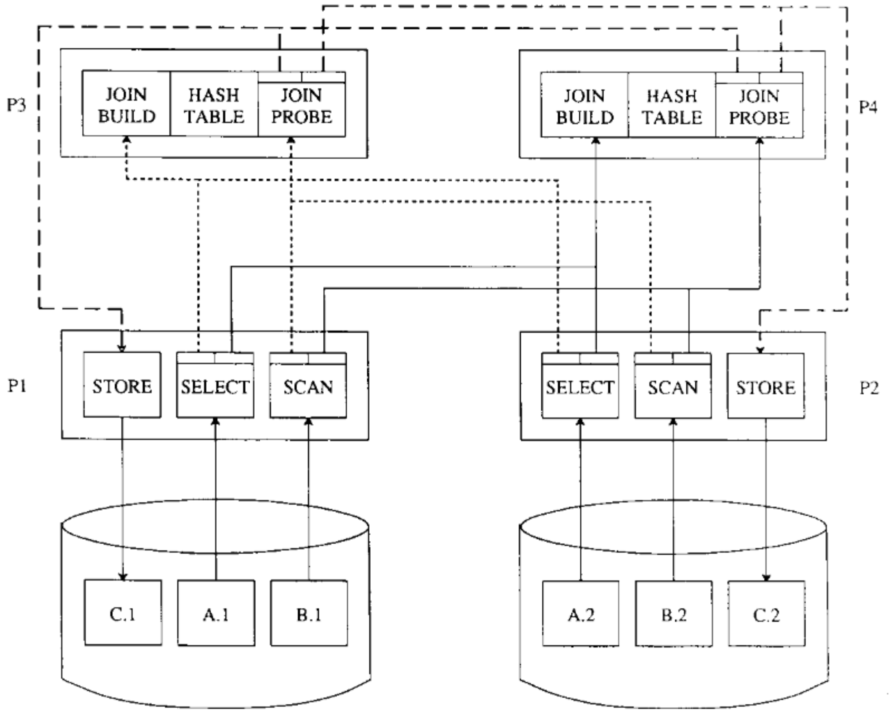
The split table defines a mapping of values to a set of destination processes.



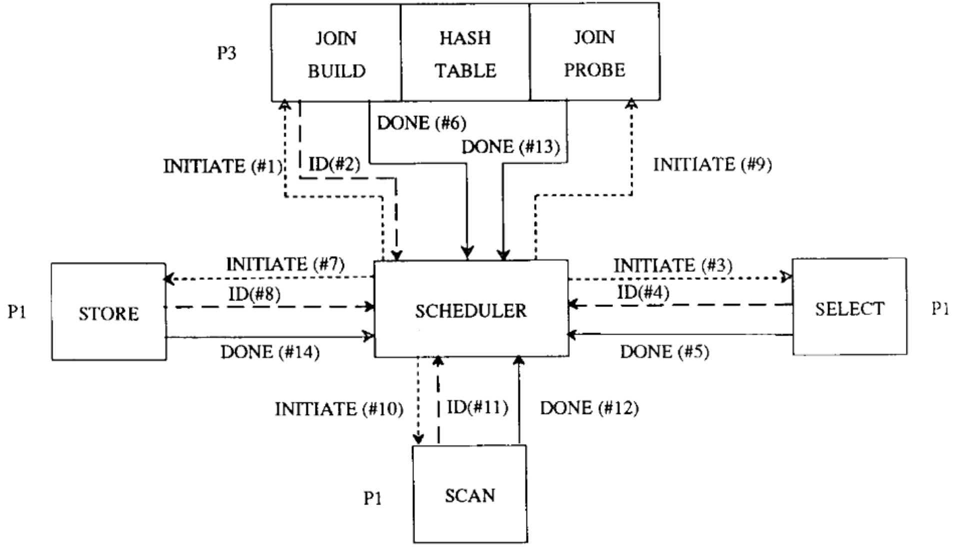
Value	Destination Process
0	(Processor #3, Port #5)
1	(Processor #2, Port #13)
2	(Processor #7, Port #6)
3	(Processor #9, Port #15)

The Parallel Simple Hash Joir





Data flow



Control flow

Query Processing

❑ Selection

- Selection on the partitioning attribute
  - Direct the selection to a subset of node if hash or range partitioned.
  - Initiate the selection on all nodes if round-robin partitioned.

❑ Join

- Partition relations into disjoint subsets (buckets) by hashing on the join attribute.
- Four types of parallel joins: sort-merge, Grace, Simple, Hybrid.
- The Hybrid hash join almost always provides the best performance.

The Parallel Hybrid Hash Join

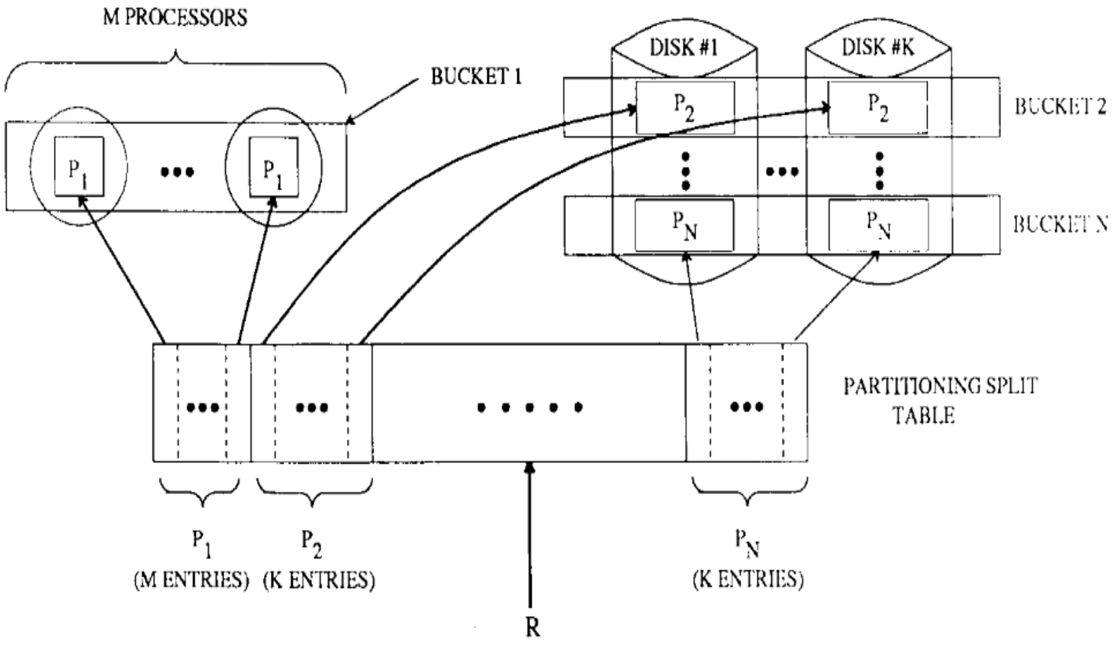


Fig. 8. Partitioning of *R* into *N* logical buckets for hybrid hash-join.

- A partitioning split table separates the relations into N logical buckets.
- A joining table sends tuples in the first bucket to M processors for the join operation.
- In-memory hash table for the first bucket of the inner table to be joined with the first bucket of the outer table.
- The N-1 buckets are temporarily stored on disks.

Query Processing Algorithms

❑ Aggregate functions

- Each processor computes a partial results on its partition.
- The processors redistribute the results on hashing on the “group by” attribute.

❑ Update operators

- Most operators are implemented with standard techniques.
- A replace operator will send a tuple to the partition to which it belongs.

Transaction and Failure Management

❑ Concurrency control

- Two-phase locking.
- A local lock manager with a lock table and a transaction wait-for-graph.
- A centralized deadlock detector communicate with each node.

❑ Recovery and Log manager

- A log record is generated when a tuple is updated.
- Log records are sent to one or more log managers.
- The log manager keeps track of the last flushed record from each node.
- The buffer managers observe the WAL protocol.

Data Placement

- Chained declustering

Node	0	1	2	3	4	5	6	7
Primary Copy	R0	R1	R2	R3	R4	R5	R6	R7
Backup Copy	r7	r0	r1	r2	r3	r4	r5	r6

- Interleaved declustering

	Cluster 0				Cluster 1			
Node	0	1	2	3	4	5	6	7
Primary Copy	R0	R1	R2	R3	R4	R5	R6	R7
Backup Copy		r0.0	r0.1	r0.2		r4.0	r4.1	r4.2
	r1.2		r1.0	r1.1	r5.2		r5.0	r5.1
	r2.1	r2.2		r2.0	r6.1	r6.2		r6.0
	r3.0	r3.1	r3.2		r7.0	r7.1	r7.2	

Load Balancing When One Node Fails

Node	0	1	2	3	4	5	6	7
Primary Copy	R0	R1	R2	R3	R4	R5	R6	R7
Backup Copy	r7	r0	r1	r2	r3	r4	r5	r6

Node	0	1	2	3	4	5	6	7
Primary Copy	R0	---	$\frac{1}{7}$R2	$\frac{2}{7}$R3	$\frac{3}{7}$R4	$\frac{4}{7}$R5	$\frac{5}{7}$R6	$\frac{6}{7}$R7
Backup Copy	$\frac{1}{7}$r7	---	r1	$\frac{6}{7}$r2	$\frac{5}{7}$r3	$\frac{4}{7}$r4	$\frac{3}{7}$r5	$\frac{2}{7}$r6

Access both the primary and backup copies to balance load on each node.

Ideal Parallelism

- **Speedup**

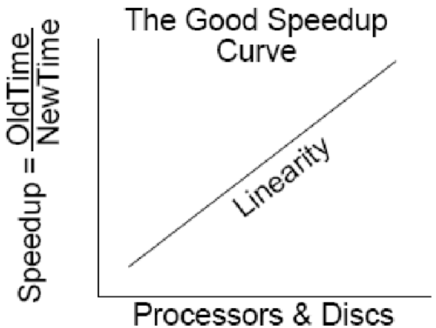
  Given a system with 1 node, does adding *n* nodes speed it up with a factor of *n* ?

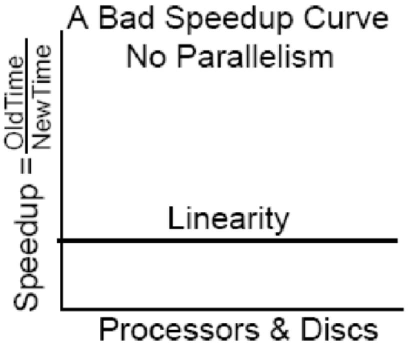
$$Speedup = \frac{small\_system\_elapsed\_time}{big\_system\_elapsed\_time}$$

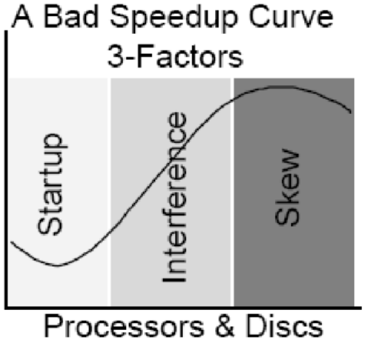
- **Scaleup**

  Given a system with 1 node, does the response time remain the same with *n* nodes ?

$$Scaleup = \frac{small\_system\_elapsed\_time\_on\_small\_problem}{big\_system\_elapsed\_time\_on\_big\_problem}$$







Discussion 2 (Groups of 4)

- The Gamma database paper is quite old (as you probably also noticed from the used hardware).
- What kind of use cases do you think did the authors have in mind?
- Why do you think parallel databases were **not** a big breakthrough at the time?
- How do you think the demand for parallel databases has changed since then?

Conclusion

- Three key ideas that enables Gamma to be scaled to hundreds of processors:
  - Horizontally partitioned relations
  - Extensive use of hash-based parallel algorithms
  - Dataflow scheduling techniques for multioperator queries

MapReduce: Simplified Data Processing on Large Clusters

Jeff Dean, Sanjay Ghemawat

Google, OSDI 2004

Slides based on those by authors and other online sources

Presenter: Tanya Prasad

Motivation

- Large scale data processing
  - Using hundreds or thousands of machines but without the hassle of management
- MapReduce benefits
  - Automatic parallelization & distribution
  - Fault tolerance
  - I/O scheduling
  - Monitoring & status updates

Programming model

- Input & Output: each a set of key/value pairs
- Programmer specifies two functions:

```
      map(in_key, in_value) -> list(out_key,
intermediate_value)
```

  - Processes each input key/value pair
  - Produces set of intermediate pairs

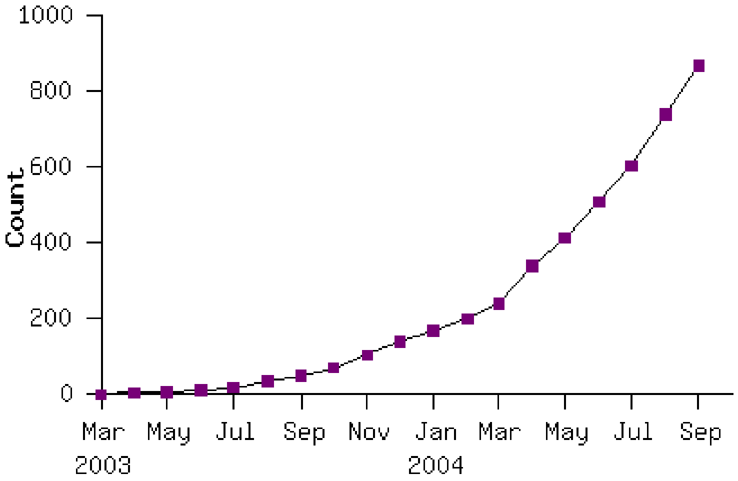
```
      reduce(out_key, list(intermediate_value)) ->
list(out_value)
```

  - Combines all intermediate values for a particular key
  - Produces a set of merged output values (usually just one)

- Inspired by similar primitives in LISP and other

MapReduce model widely applicable

- MapReduce programs in Google source tree (2003-04)



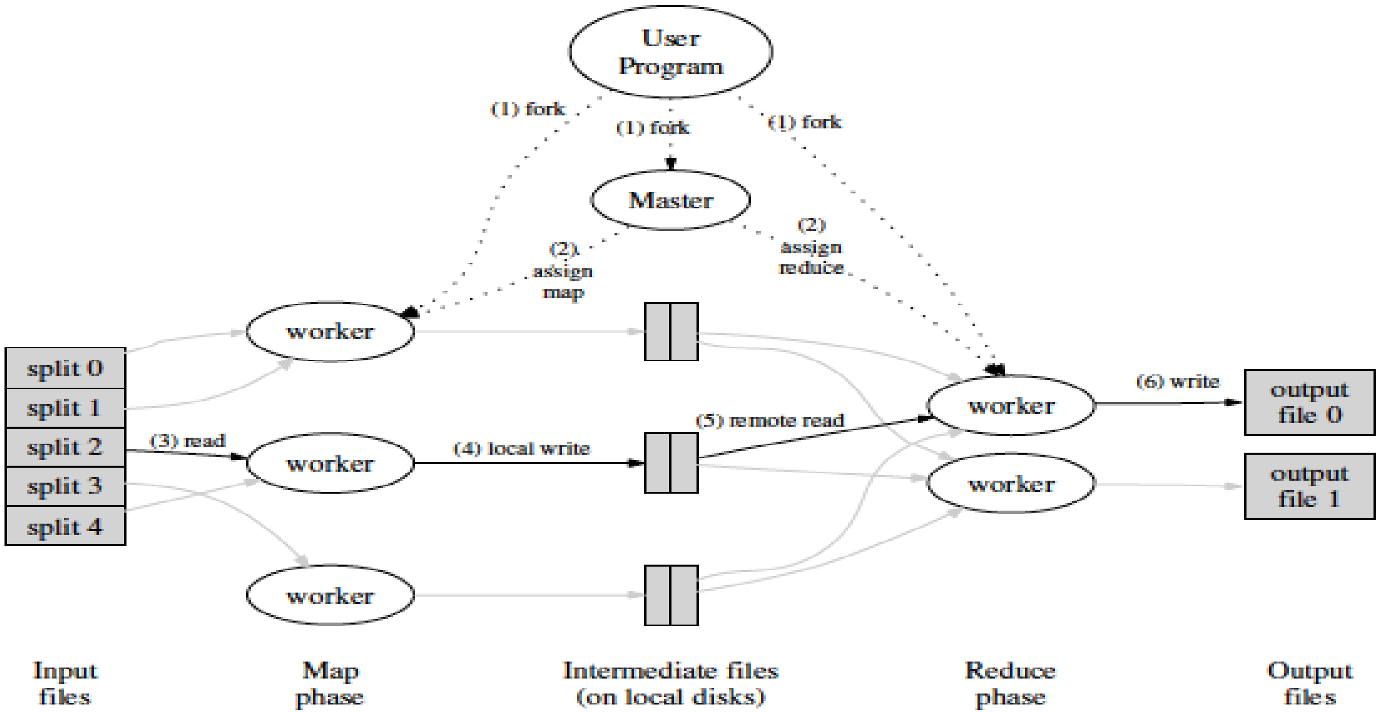
Examples

distributed grep	distributed sort	web link-graph reversal
term-vector / host	web access log stats	inverted index construction
document clustering	machine learning	statistical machine translation
...	...	...

Implementation overview

- Typical cluster:
  - 100s/1000s of 2-CPU x86 machines, 2-4 GB of memory
  - Limited bisection bandwidth
  - Storage is on local IDE disks
  - GFS: distributed file system manages data (SOSP'03)
  - Job scheduling system: jobs made up of tasks, scheduler assigns tasks to machines
- Implementation as C++ library linked into user programs

Overall execution workflow



Discussion 3 (Pairs)

- MapReduce breaks with a lot of conventions: Input data has no schema, programs are written in Java, no indices,...
  - Why do you think MapReduce was still such a huge success?
  - Why or why not is that surprising to you?
- Discuss the questions with the lessons from last week's discussion in mind. How do they hold up here?
  - Lesson 12: Unless there is a big performance or functionality advantage, new constructs will go nowhere
  - Lesson 13: Packages will not sell to users unless they are in “major pain”
  - Lesson 16: Schema-last is probably a niche market

Fault-tolerance via re-execution

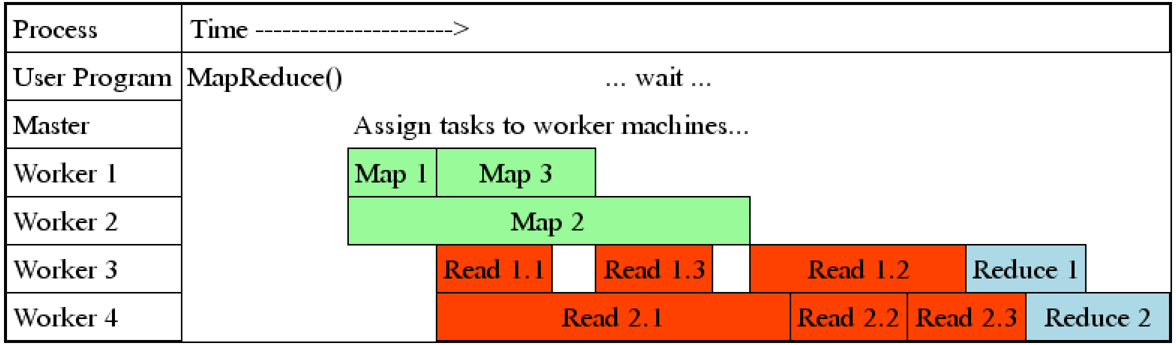
- On worker failure:
  - Detect failure via periodic heartbeats
  - Re-execute completed and in-progress *map* tasks
    - Output stored on the local disk becomes inaccessible
  - Re-execute in progress *reduce* tasks
    - Output stored in a global file system
  - Task completion committed through master
- Master failure:
  - Left unhandled as considered unlikely
  - Abort the MapReduce computation

Locality Optimization

- Master scheduling policy:
  - Asks GFS for locations of replicas of input file blocks
  - Map tasks typically split into 64MB (== GFS block size)
  - Map tasks scheduled so GFS input block replica are on same machine or same rack or nearest machine.
  - Goal to reduce communication overhead as much as possible
- Effect: Thousands of machines read input at local disk speed
  - Without this, rack switches limit read rate

Task Granularity

- Fine granularity tasks: map tasks >> machines
  - Minimizes time for fault recovery
  - Can pipeline shuffling with map execution
  - Better dynamic load balancing
- Often use 200K map and 5000 reduce tasks running on 2000 machines



Backup Execution

- Slow workers significantly lengthen completion time
  - Other jobs consuming resources on machine
  - Bad disks with soft errors transfer data very slowly
  - Weird things: processor caches disabled (!!)
- Solution: Near end of phase, spawn backup task copies
  - Whichever one finishes first "wins"
- Benefit: Dramatically shortens job completion time

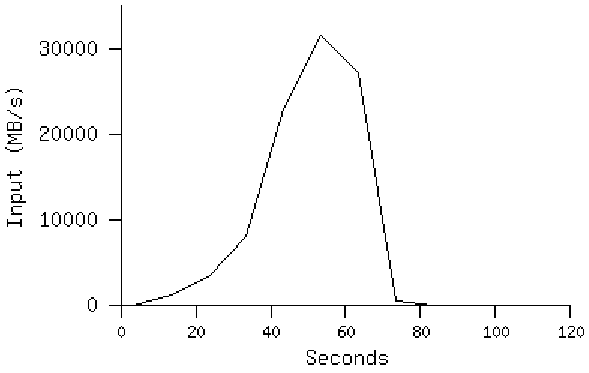
Skipping Bad Records

- Map/Reduce functions sometimes fail for particular inputs
  - Best solution is to debug & fix, but not always possible
- On segmentation fault:
  - Send UDP packet to master from the signal handler
  - Include sequence number of record being processed
- If master sees two failures for the same record:
  - Next worker is told to skip the record
- Effect: Can work around bugs in third-party libraries

Some Refinements

- Sorting guarantees within each reduce partition
- Compression of intermediate data
- Combiner: useful for saving network bandwidth
- Local sequential execution for debugging/testing
- User-defined counters

MapReduce Grep

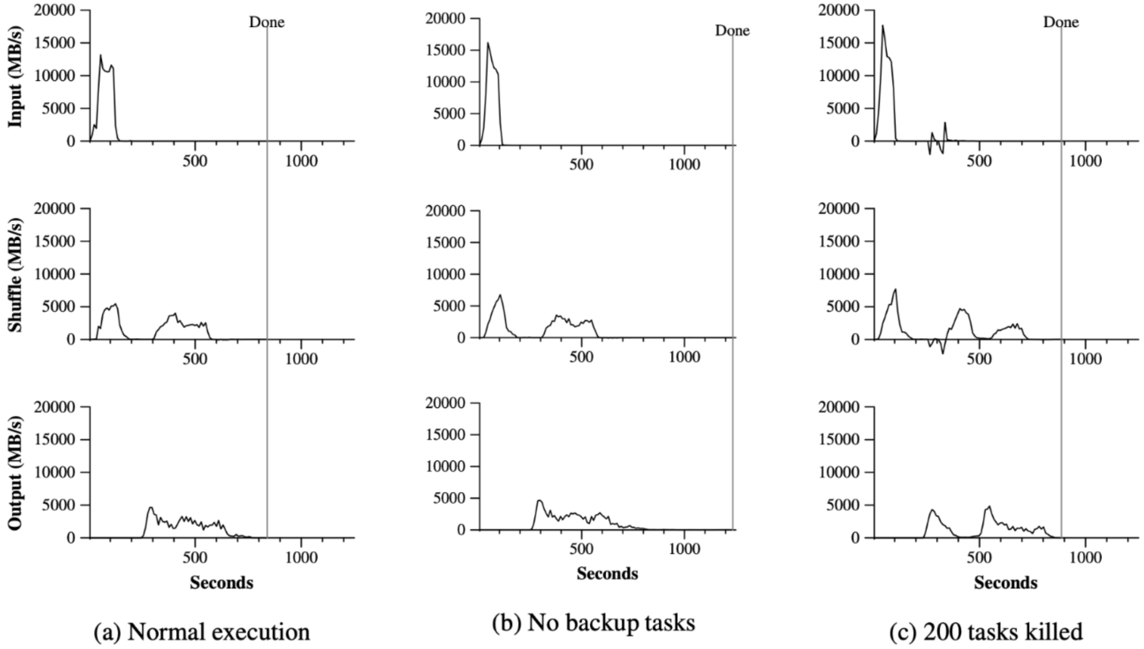


Locality optimization helps:

- 1800 machines read 1 TB at peak ~31 GB/s
- W/out this, rack switches would limit to 10 GB/s

Startup overhead is significant for short jobs

MapReduce Sort



- Backup tasks reduce job completion time a lot!
- System deals well with failures

Google Experience: Rewrite of Production Indexing System

- Rewrote Google's production indexing system using MapReduce
  - New code is simpler, easier to understand
  - MapReduce takes care of failures, slow machines
  - Easy to make indexing faster by adding more machines

Discussion 4 (Groups of 4)

- With the Gamma database project and MapReduce we have seen two models to parallelize data processing:
  - What are the differences and similarities?
  - Which use cases are they designed for? Do they have the same kind of applications in mind?
  - Which model do you find more convincing and why?

- Gamma Database key features:
  - Parallel Database
  - Horizontally partitioned relations
  - Extensive use of hash-based parallel algorithms
  - Dataflow scheduling techniques for multioperator queries

Conclusions

- MapReduce has proven to be a useful abstraction.
- Network bandwidth is a scarce resource.
- Redundant execution can reduce the impact of slow machines and machine failures.