From Shared Virtual Memory to Parameter Servers

Phil Gibbons

15-712 F15

Lecture 16

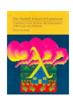
Today's Reminders

- **No Class Friday or Monday**
- **My office hours: Today after class**

Memory Coherence in Shared Virtual Memory Systems

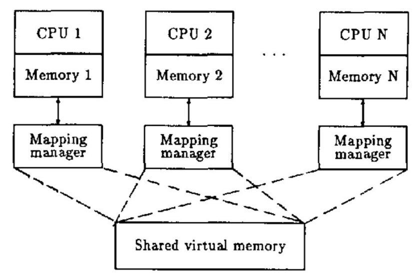
[PODC'86, TOCS 1989]

- **Kai Li** (Princeton)
  - datadomain co-founder (acquired for $2.4B !)
  - ACM/IEEE Fellow; NAE
- **Paul Hudak** (Yale, d. 4/15)
  - ACM Fellow; Co-designed Haskell



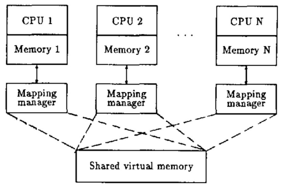
"The paper shows how to simulate coherent shared memory on a cluster, and also introduces directory-based distributed cache-coherence. It spawned an entire research area, and introduced cache coherence mechanisms that are widely used in industry." – SigOps HoF citation

Shared Virtual Memory



- **Page data between the physical memories of the processors (as well as between physical memory & disk)**
  - Common mechanism for both
  - Once page in, data access is familiar read/write

Memory Coherence Problem



- **Memory coherence:** **read returns value of most recent write to same address**
- **Differs from multiprocessor cache coherence**
  - Small caches, fast bus, done in HW => write conflicts incur small delay

"Both theoretical & practical results show MC problem can be solved efficiently on a loosely coupled multiprocessor"

Why Shared Virtual Memory Should Have Good Performance

- **Unshared data is fine**
- **Read-only shared data is fine**
- **Updates to shared data?**
  - Each individual thread has good locality in its writes
  - Common goal in designing parallel algorithms is to minimize write contention among threads

Granularity

- **Why larger pages?**
  - Amortize communication overheads
  - 1000s of bytes roughly same cost as 10s of bytes
- **Why smaller pages?**
  - Minimizes chance for contention (false sharing)
- **Choose size to match existing VM page size**
  - Can use existing page protection mechanisms: single instructions will trigger page faults & trap to handlers, e.g. to enforce memory coherence mechanisms

Maintaining Coherence

- **Each page can be**
  - Read-shared by 1 or more processors, or
  - Exclusively owned by a processor who can write
- **Directory-based coherence ("Fixed Distributed Manager")**
  - Management of pages is partitioned across processors
  - On fault, consult manager for the page
  - Manager tracks set of read-sharers ("copyset") or exclusive owner ("owner") & serves as point of serialization

  [Run through example on board]
- **Works well for Cache-coherence.**
  What's the issue for Shared Virtual Memory?
  - Want to avoid extra hop to directory/manager

Dynamic Distributed Manager: Metadata on pages

- **Ptable[p].access = {read, write, nil}**
- **Ptable[p].copyset = processors with read copies**
- **Ptable[p].lock**
- **Ptable[p].probOwner = likely owner**

Dynamic Distributed Manager

```
Read-fault handler:
  Lock(PTable[p].lock);
  ask PTable[p].probOwner for read access to p;
  PTable[p].probOwner := ReplyNode;
  PTable[p].access := read;
  Unlock(PTable[p].lock);
Read server:
  Lock(PTable[p].lock);
  IF I am owner THEN BEGIN
    PTable[p].copyset
      := PTable[p].copyset ∪ {RequestNode};
    PTable[p].access := read;
    send p to RequestNode;
    END
  ELSE BEGIN
    forward request to PTable[p].probOwner;
    PTable[p].probOwner := RequestNode;
    END;
  Unlock(PTable[p].lock);
```


Dynamic Distributed Manager

```
Write fault handler:
  Lock( PTable[ p ].lock );
  ask PTable[ p ].probOwner for write access to page p;
  Invalidate( p, PTable[ p ].copyset );
  PTable[ p ].probOwner := self;
  PTable[ p ].access := write;
  PTable[ p ].copyset := {};
  Unlock( PTable[ p ].lock );

Write server:
  Lock( PTable[ p ].lock );
  IF I am owner THEN BEGIN
      PTable[ p ].access := nil;
      send p and PTable[ p ].copyset;
      PTable[ p ].probOwner := RequestNode;
      END
  ELSE BEGIN
      forward request to PTable[ p ].probOwner;
      PTable[ p ].probOwner := RequestNode;
      END;
  Unlock( PTable[ p ].lock );
```


Speedups for 3D PDE

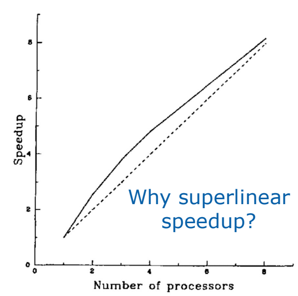


Fig. 5. Speedups of a 3-D PDE where $n = 50^3$.

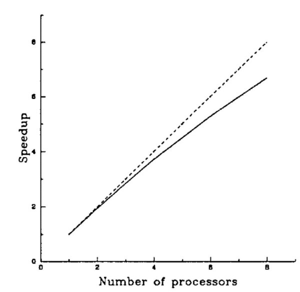


Fig. 7. Speedups of a 3-D PDE where $n = 40^3$.

Note: Static mapping (CM*) only comes close to SVM with heroic programming effort

Speedups for Sort & Dot-Product

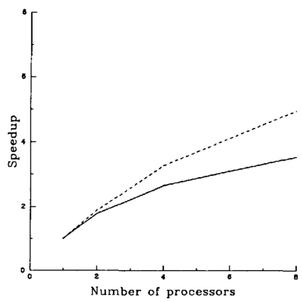


Fig. 8. Speedup of the merge-split sort.

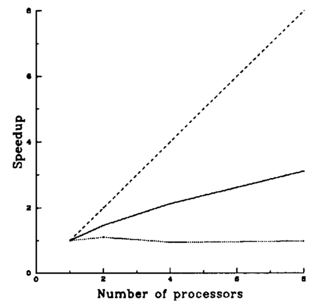


Fig. 9. Speedup of the dot-product program.

Coherence Algorithms

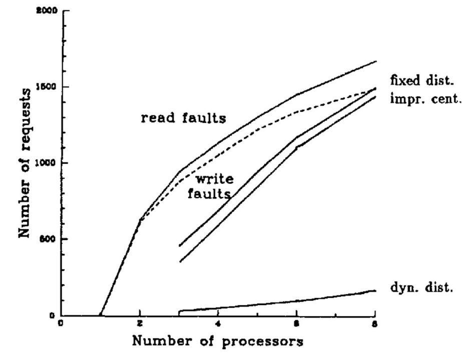


Fig. 11. Forwarding requests.

3D PDE
probOwner field usually correct; page sharing is small

Limitations

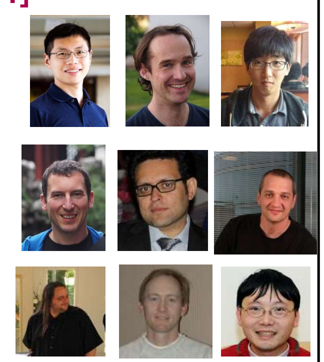
- **Main classes of programs that would perform poorly:**
  - Frequent updates to shared data
  - Excessively large data sets that are only read once
- **Only ran on up to 8 processors**

What should this paper get credit for?

Scaling Distributed Machine Learning with the Parameter Server

[OSDI'14]

- **Mu Li** (CMU)
- **David Andersen** (CMU)
- **Jun Woo Park** (CMU)
- **Alexander Smola** (CMU)
- **Amr Ahmed** (Google)
- **Vanja Josifovski** (Pinterest)
- **James Long** (Google)
- **Eugene Shekita** (Google)
- **Bor-Yiing Su** (Google)

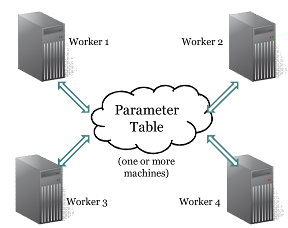


Some Big Learning Frameworks

- **GraphLab (Dato)**
  - Carlos Guestrin (CMU->Washington)
- **Spark (Databricks)**
  - Ion Stoica (UC Berkeley)
- **Petuum**
  - Eric Xing, Greg Ganger, Phil Gibbons, Garth Gibson (CMU)
- **Parameter Server (Marianas Labs)**
  - Alex Smola, Dave Andersen (CMU)

Parameter Servers for Distributed ML

- **Provides all machines with convenient access to global model parameters**
- **Enables easy conversion of single-machine parallel ML algorithms**
  - "Distributed shared memory" programming style
  - Replace local memory access with PS access



Single Machine Parallel

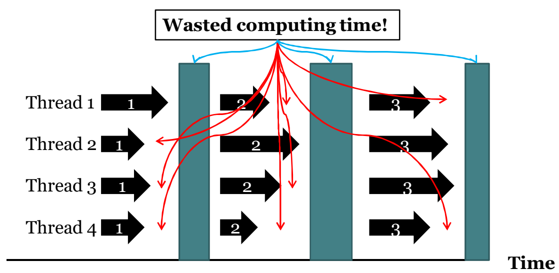
```
UpdateVar(i) {
  old = y[i]
  delta = f(old)
  y[i] += delta
}
```

Distributed with PS

```
UpdateVar(i) {
  old = PS.read(y,i)
  delta = f(old)
  PS.inc(y,i,delta)
}
```

† Ahmed et al. (WSDM 2012), Power and Li (OSDI 2010)

The Cost of Bulk Synchrony

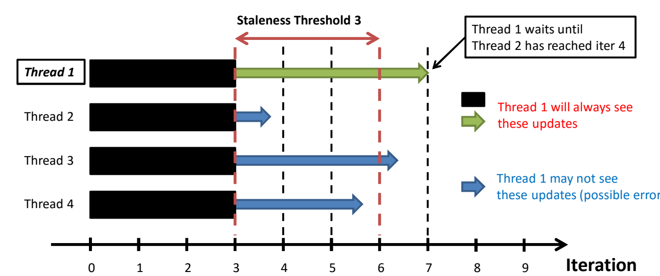


Threads must wait for each other
End-of-iteration sync gets longer with larger clusters

Precious computing time wasted

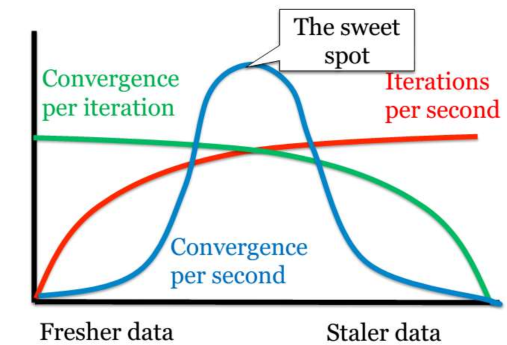
But: Fully asynchronous => No algorithm convergence guarantees

Stale Synchronous Parallel (SSP)



Allow threads to *usually* run at own pace
Fastest/slowest threads not allowed to drift >S iterations apart
Protocol: check cache first; if too old, get latest version from network
Consequence: fast threads must check network every iteration
Slow threads check only every S iterations – fewer network accesses, so catch up!

Staleness Sweet Spot

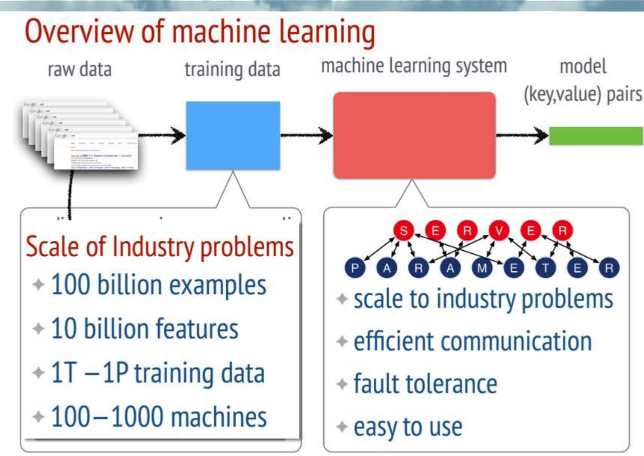


[ATC'14]

Enhancements to SSP

- **Early transmission of larger parameter changes, up to bandwidth limit**
- **Find sets of parameters with weak dependency to compute on in parallel**
  - Reduces errors from parallelization
- **Low-overhead work migration to eliminate transient straggler effects**
- **Exploit repeated access patterns of iterative algorithms** (IterStore)
  - Optimizations: prefetching, parameter data placement, static cache policies, static data structures, NUMA memory management

Parameter Server



Distributed Data Analysis Systems

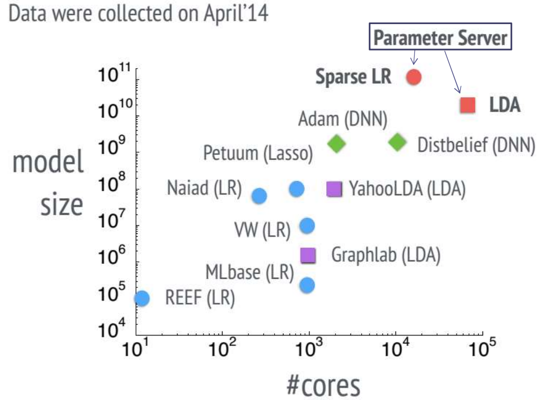
	Shared Data	Consistency	Fault Tolerance
Graphlab [34]	graph	eventual	checkpoint
Petuum [12]	hash table	delay bound	none
REEF [10]	array	BSP	checkpoint
Naiad [37]	(key,value)	multiple	checkpoint
Mlbase [29]	table	BSP	RDD
Parameter Server	(sparse) vector/matrix	various	continuous

Fair characterizations?

Parameter Server

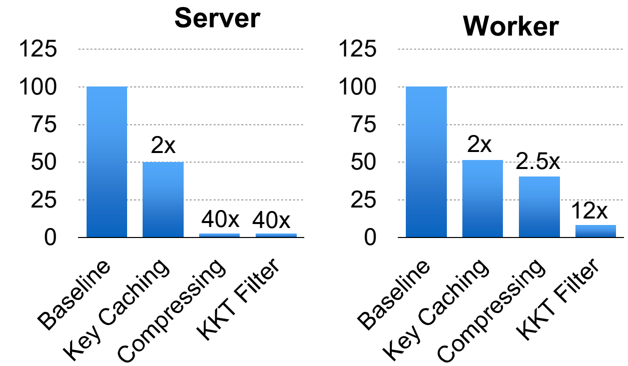
Largest experiments of related systems

Data were collected on April'14

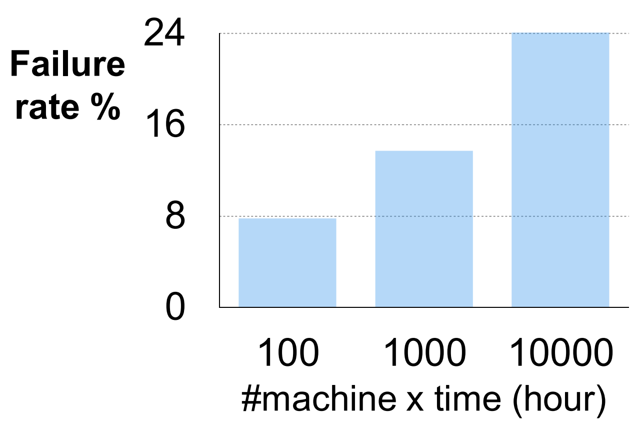


Traffic Reduction by Filters

Ad click prediction
636TB data, 1TB model, and 1000 machines



Machine learning job logs in a 3-month period

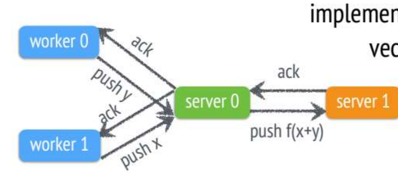


Fault Tolerance

- Model is partitioned by consistent hashing
- Default replication: Chain replication (consistent, safe)

- Option: Aggregation reduces backup traffic (algo specific)

implemented by efficient vector clock



Next Wednesday’s Papers

Application Performance and Flexibility on Exokernel Systems

Frans Kaashoek, Dawson Engler, Greg Ganger, Hector Briceno, Russell Hunt, David Mazzieres, Thomas Pinckney, Robert Grimm, John Jannotti, Kenneth Mackenzie

SOSP’97

Safe Kernel Extensions without Run-Time Checking

George Necula and Peter Lee

SigOps HoF paper