

Google Base Infrastructure:

GFS and MapReduce

Johannes Passing

Agenda

- Context
- Google File System
 - Aims
 - Design
 - Walk through
- MapReduce
 - Aims
 - Design
 - Walk through
- Conclusion

Context

- Google's scalability strategy
 - Cheap hardware
 - Lots of it
 - More than 450.000 servers (NYT, 2006)
- Basic Problems
 - Faults are the norm
 - Software must be massively parallelized
 - Writing distributed software is hard

GFS: Aims

- Aims
 - Must scale to thousands of servers
 - Fault tolerance/Data redundancy
 - Optimize for large WORM files
 - Optimize for large reads
 - Favor throughput over latency
- Non-Aims
 - Space efficiency
 - Client side caching futile
 - POSIX compliance

→ Significant deviation from standard DFS (NFS, AFS, DFS, ...)

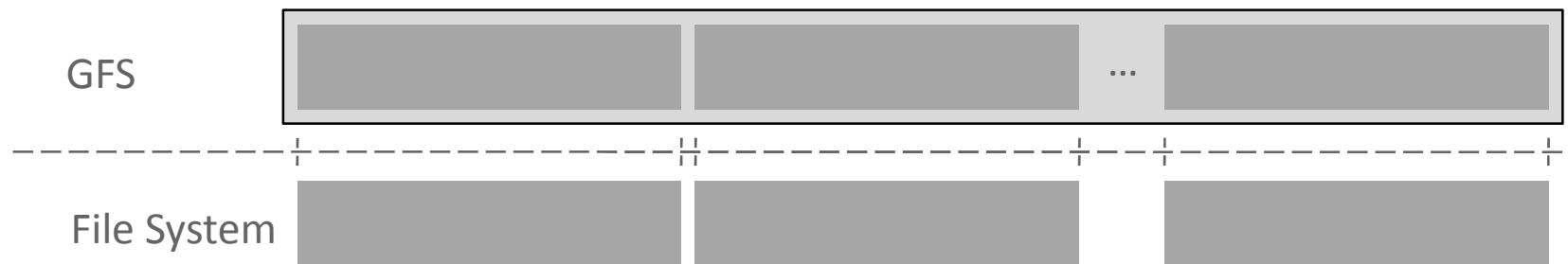
Design Choices

- File system cluster
 - Spread data over thousands of machines
 - Provide safety by redundant storage
 - Master/Slave architecture

Design Choices

- Files split into chunks
 - Unit of management and distribution
 - Replicated across servers
 - Subdivided into blocks for integrity checks

→ Accept internal fragmentation for better throughput



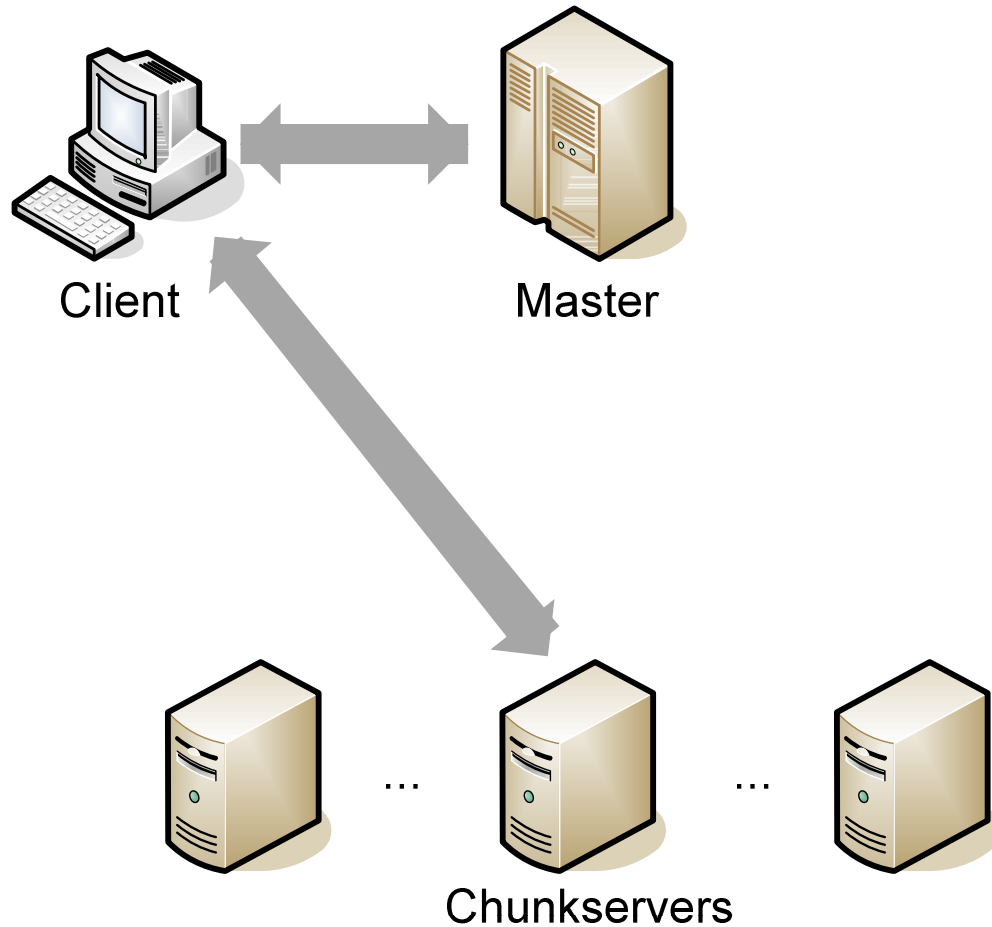
Architecture: Chunkserver

- Host chunks
 - Store chunks as files in Linux FS
 - Verify data integrity
 - Implemented entirely in user mode
- Fault tolerance
 - Fail requests on integrity violation detection
 - All chunks stored on at least one other server
 - Re-replication restores server after downtime

Architecture: Master

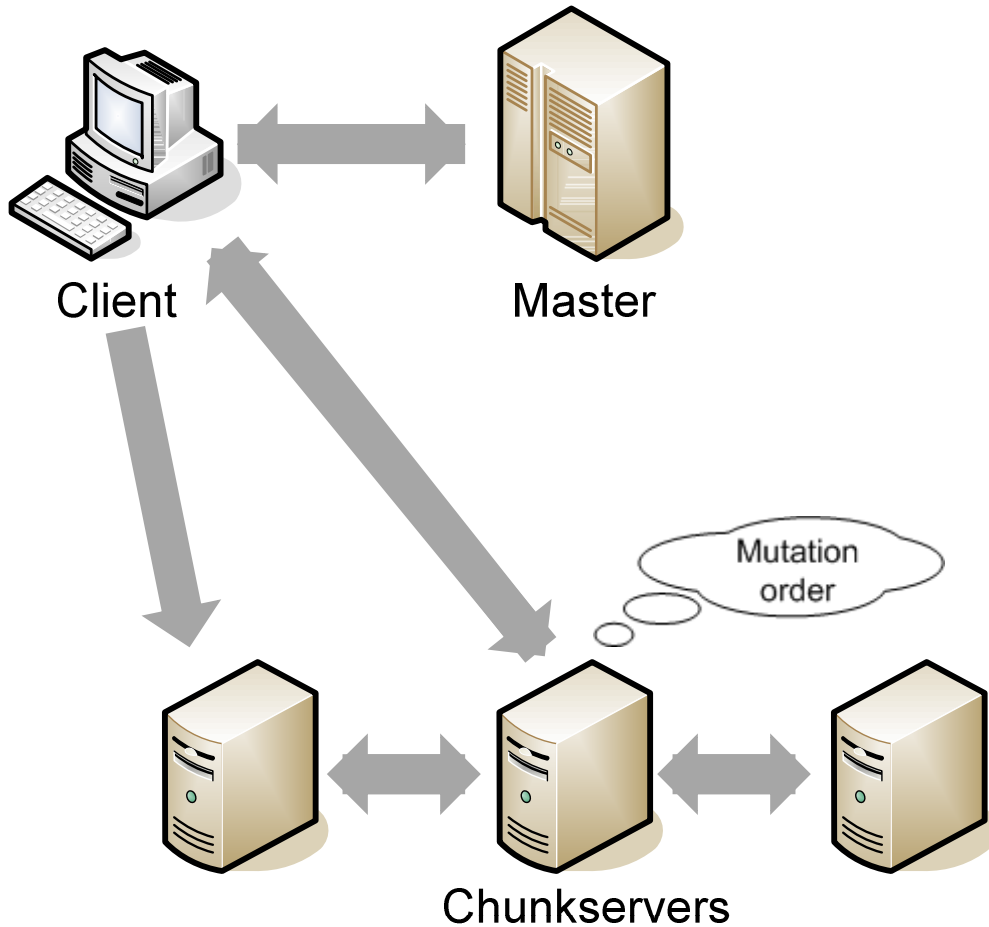
- Namespace and metadata management
- Cluster management
 - Chunk location registry (transient)
 - Chunk placement decisions
 - Re-replication, Garbage collection
 - Health monitoring
- Fault tolerance
 - Backed by shadow masters
 - Mirrored operations log
 - Periodic snapshots

Reading a chunk



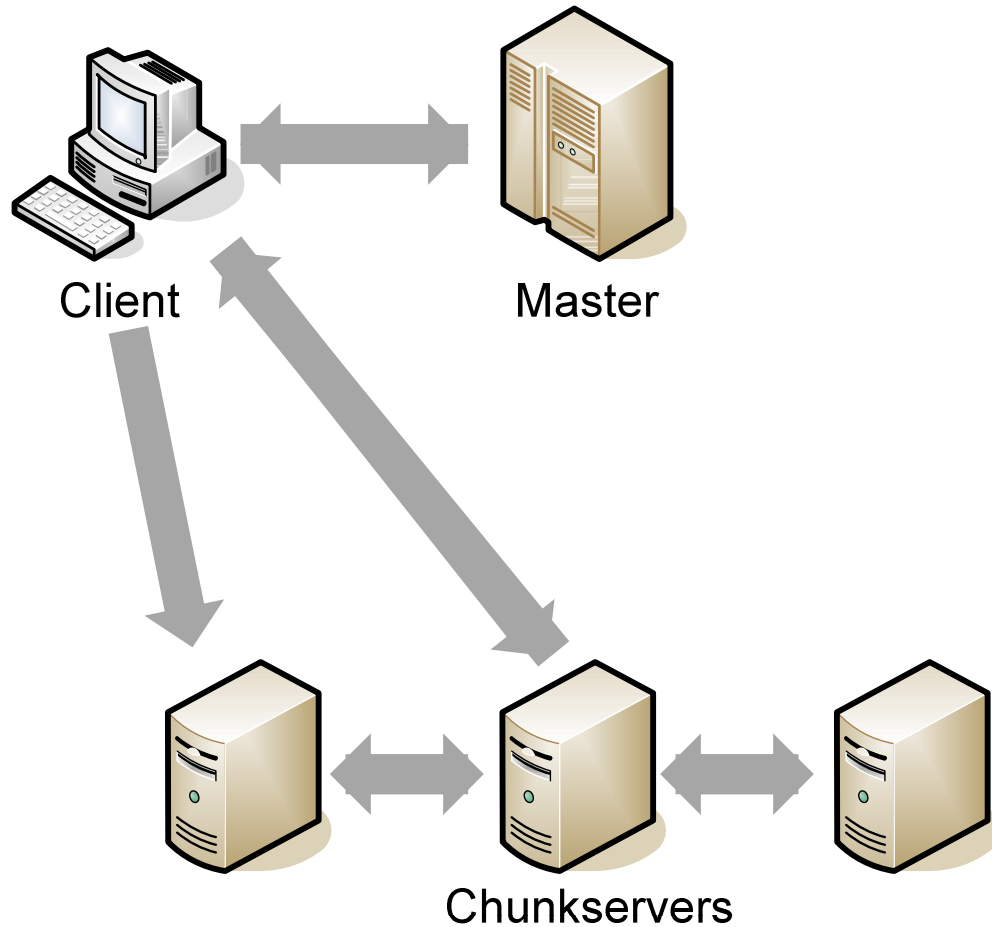
- Send filename and offset
- Determine file and chunk
- Return chunk locations and version
- Choose closest server and request chunk
- Verify chunk version and block checksums
- Return data if valid, else fail request

Writing a chunk



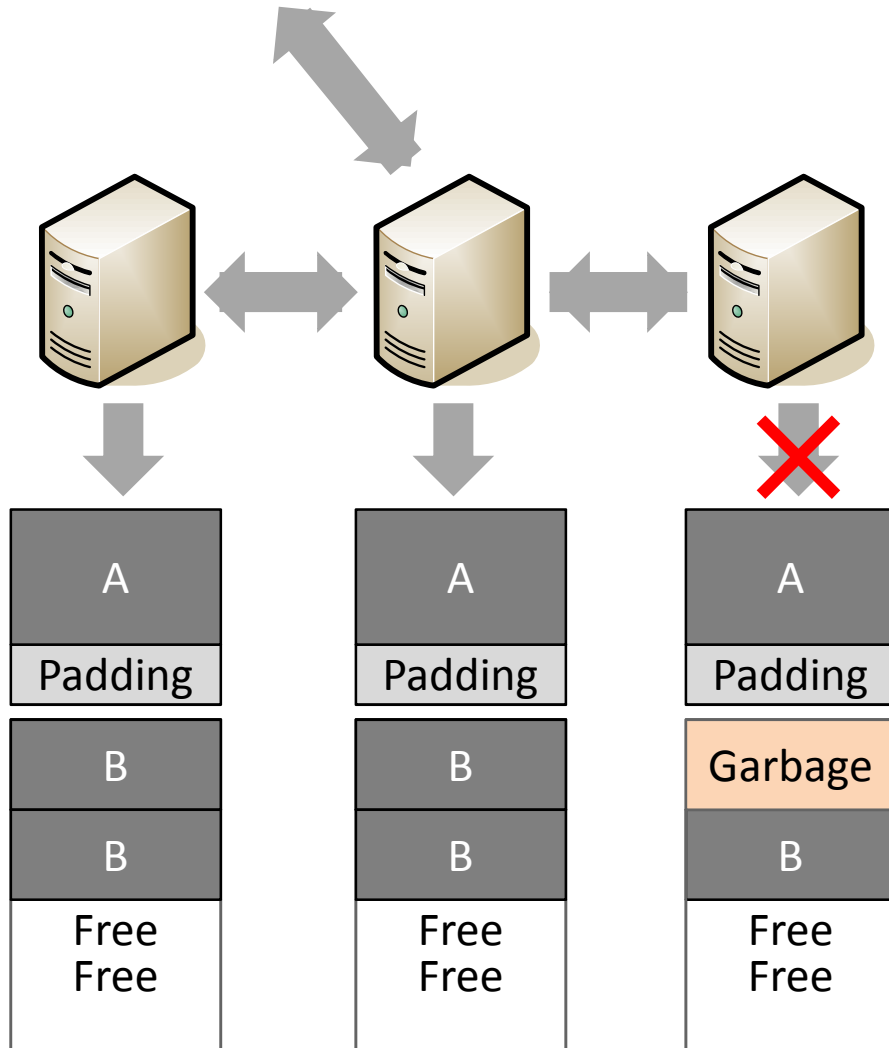
- Send filename and offset
- Determine lease owner/
grant lease
- Return chunk locations
and lease owner
- Push data to *any* chunkserver
- Forward along replica chain
- Send write request to primary
- Create mutation order, apply
- Forward write request
- Validate blocks, apply
modifications , inc version, ack
- Reply to client

Appending a record (1/2)



- Send append request
- Choose offset
- Return chunk locations and lease owner
- Push data to *any* chunkserver
- Forward along replica chain
- Send append request
- Check for left space in chunk
- Pad chunk and request retry
- Request chunk allocation
- Return chunk locations and lease owner

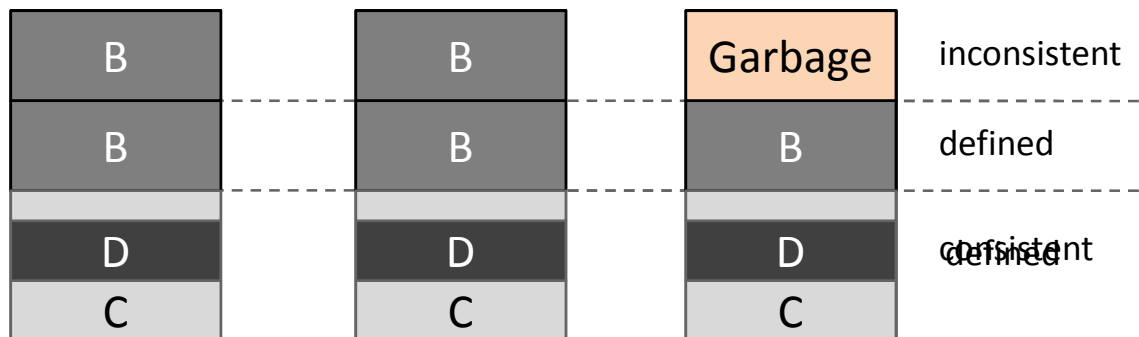
Appending a record (2/2)



- Send append request
- Allocate chunk, write data
- Forward request to replicas
- Write data, failure occurs
- Fail request
- Client retries
- Write data
- Forward request to replicas
- Write data
- Success

File Region Consistency

- File region states:
 - Consistent \Leftrightarrow All clients see same data
 - Defined \Leftrightarrow Consistent \wedge clients see change in its entirety
 - Inconsistent
- Consequences
 - Unique record IDs to identify duplicates
 - Records should be self-identifying and self-validating



Fault tolerance GFS

- Automatic re-replication
- Replicas are spread across racks

- Fault mitigations
 - Data corruption → choose different replica
 - Machine crash → choose replica on different machine
 - Network outage → choose replica in different network
 - Master crash → shadow masters

Wrap-up GFS

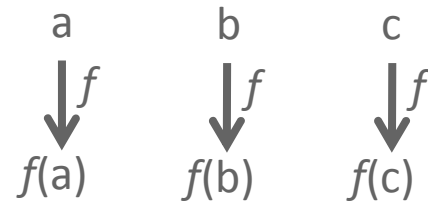
- Proven scalability, fault tolerance and performance
- Highly specialized
- Distribution transparent to clients
- Only partial abstraction – clients must cooperate
- Network is the bottleneck

MapReduce: Aims

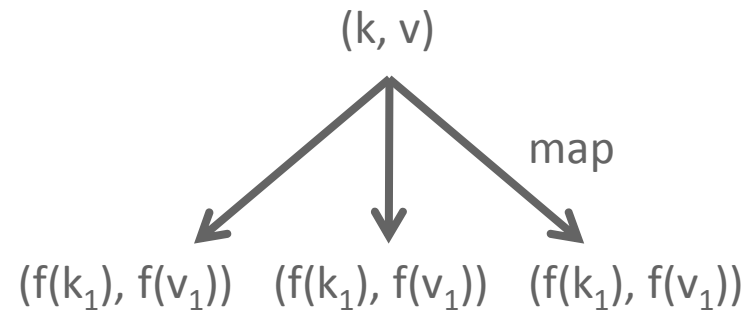
- Aims
 - Unified model for large scale distributed data processing
 - Massively parallelizable
 - Distribution transparent to developer
 - Fault tolerance
 - Allow moving of computation close to data

Higher order functions

- Functional: $\text{map}(f, [a, b, c, \dots])$

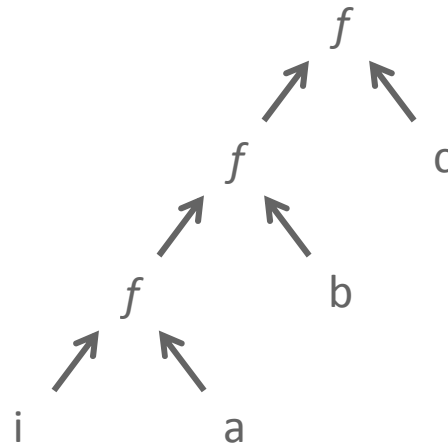


- Google: $\text{map}(k, v)$

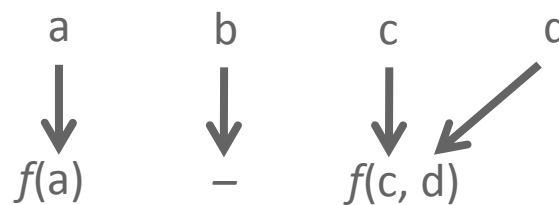


Higher order functions

- Functional: $\text{foldl}/\text{reduce}(f, [a, b, c, \dots], i)$



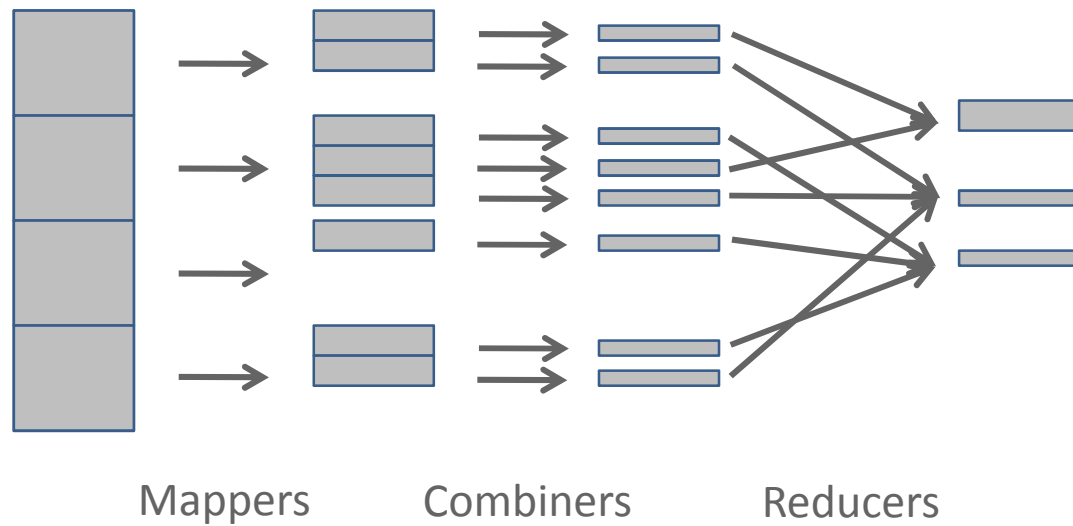
- Google: $\text{reduce}(k, [a, b, c, d, \dots])$



Idea

- Observation
 - Map can be easily parallelized
 - Reduce *might* be parallelized
- Idea
 - Design application around map and reduce scheme
 - Infrastructure manages scheduling and distribution
 - User implements map and reduce
- Constraints
 - All data coerced into key/value pairs

Walkthrough



- Spawn master
- Provide data
 - e.g. from GFS or BigTable
- Create M splits
- Spawn M mappers
- Map
 - Once per key/value pair
 - Partition into R buckets
- Spawn up to $R * M$ combiners
- Combine
- Barrier
- Spawn up to R reducers
- Reduce

Scheduling

- Locality
 - Mappers scheduled close to data
 - Chunk replication improves locality
 - Reducers run on same machine as mappers
- Choosing M and R
 - Load balancing & fast recovery vs. number of output files
 - $M \gg R$, R small multiple of #machines
- Schedule backup tasks to avoid *stragglers*

Fault tolerance MapReduce

- Fault mitigations
 - Map crash
 - All intermediate data in questionable state
 - Repeat *all* map tasks of machine
 - Rationale of barrier
 - Reduce crash
 - Data is global, completed stages marked
 - Repeat *crashed* reduce task only
 - Master crash
 - start over
 - Repeated crashes on certain records
 - Skip records

Wrap-up MapReduce

- Proven scalability
- Restrict programming model for better runtime support
- Tailored to Google's needs
- Programming model is fairly low-level

Conclusion

- Rethinking infrastructure
 - Consequent design for scalability and performance
 - Highly specialized solutions
 - Not generally applicable
 - Solutions more low-level than usual
- Maintenance efforts may be significant

„We believe we get tremendous competitive advantage by essentially building our own infrastructures“

--Eric Schmidt

References

- Jeffrey Dean and Sanjay Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters”, 2004
- Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung, “The Google File System”, 2003
- Ralf Lämmel, “Google's MapReduce Programming Model – Revisited”, SCP journal, 2006
- Google, “Cluster Computing and MapReduce”, <http://code.google.com/edu/content/submissions/mapreduce-minilecture/listing.html>, 2007
- David F. Carr, “How Google Works”, <http://www.baselinemag.com/c/a/Projects-Networks-and-Storage/How-Google-Works/>, 2006
- Colby Ranger, Ramanan Raghuraman, Arun Penmetsa, Gary Bradski, Christos Kozyrakis, “Evaluating MapReduce for Multi-core and Multiprocessor Systems”, 2007