Optimizing Distributed Read-Only Transactions Using Multiversion Concurrency

Dan Ports Austin Clements Irene Zhang

Tuesday, December 11, 2007

Distributed Transactions

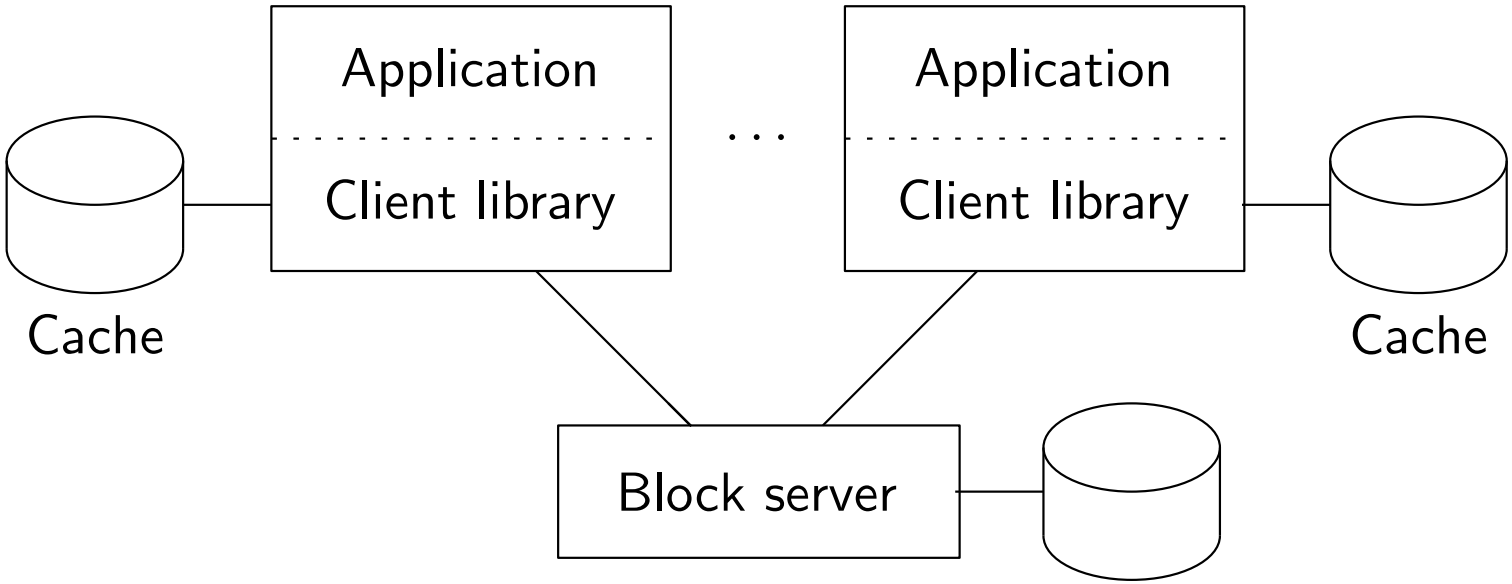
Distributed transactions are useful for many applications

Distributed Transactions

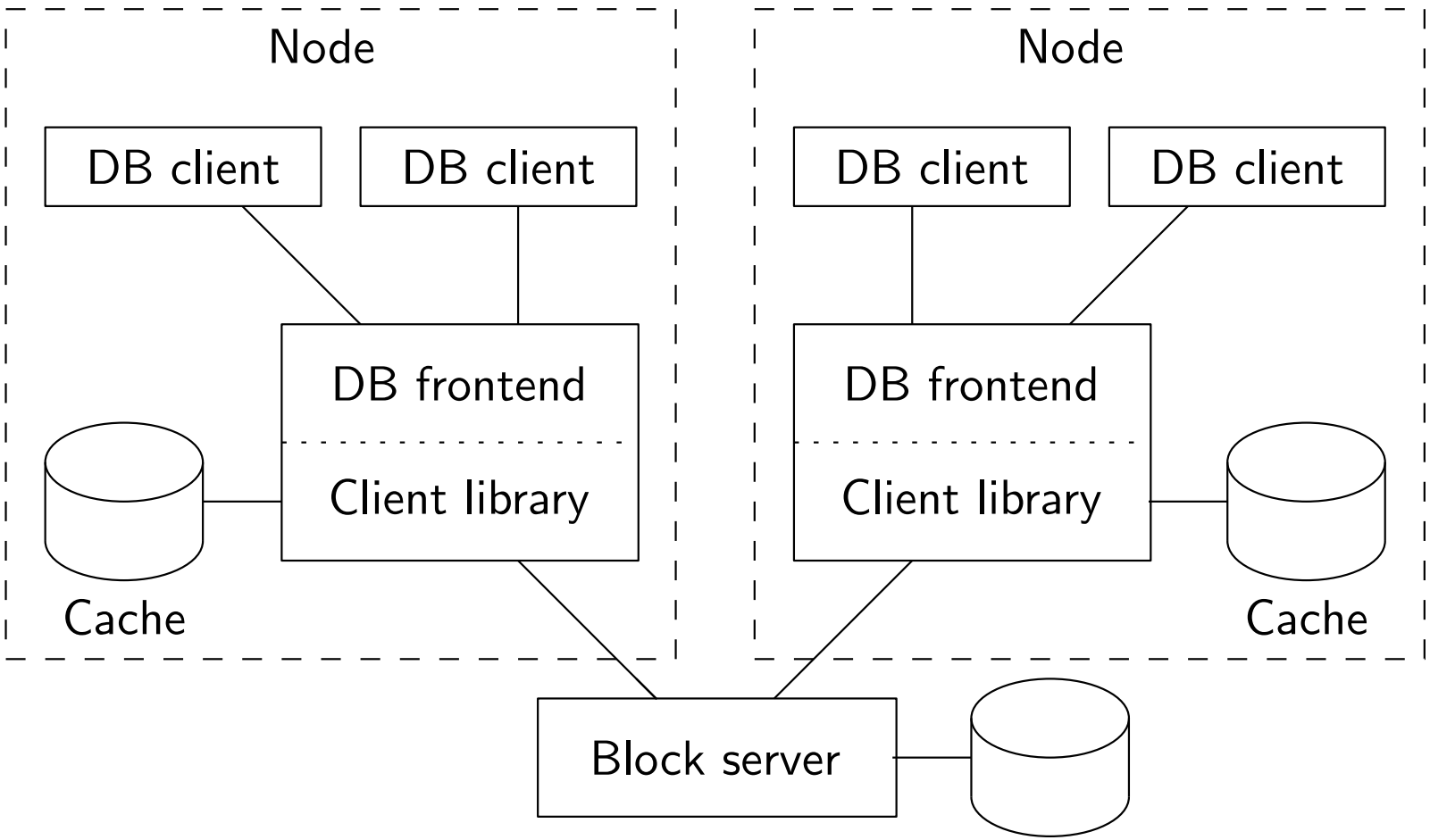
Distributed transactions are useful for many applications

...but slow

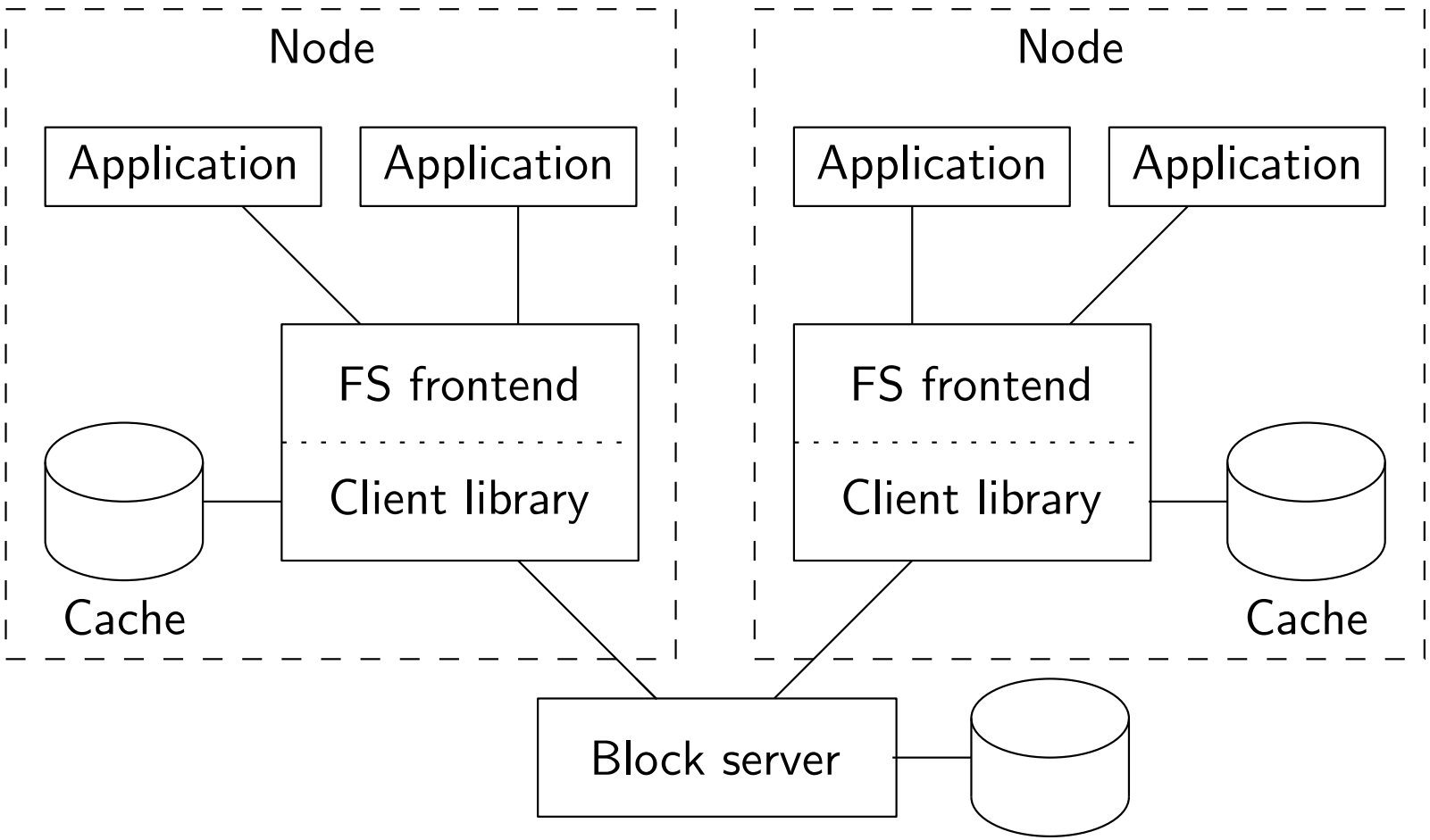
System Architecture



System Architecture



System Architecture



Distributed Transactions Can Be Fast

Lower isolation levels?

- *e.g.* `READ COMMITTED`, snapshot isolation, ...
- any hope for correctness? sanity?

Distributed Transactions Can Be Fast

Lower isolation levels?

- *e.g.* `READ COMMITTED`, snapshot isolation, ...
- any hope for correctness? sanity?

Our solution:

weaken *causality* instead of *serializability*

- All operations transactionally consistent
- Read only transactions may run slightly in past

Properties

1. **Serializability**
2. ϵ-**Freshness**
3. r/o transactions **do not block or abort**
4. **Local Causality**

Properties

1. **Serializability**
2. ϵ-**Freshness**
3. r/o transactions **do not block or abort**
4. **Local Causality**

Anomaly: acausality

A read-only transaction may not see the results of a transaction that just committed on another node.

Performance

- Built `ext2`-like filesystem atop block store
- Replayed 20,000 operations over 13 minutes from Berkeley NFS server trace
- 116 parallel clients
- Inferred transactions (open-close)
- 2 second allowable staleness
- Compared against standard OCC

Performance

	Plain OCC	Read-Opt.	Improvement
Network	15.0 MB	11.0 MB	27%
Aborts	392	22	94%
CPU time	14.5 min.	35 sec.	96%