FT-MPI, Fault-Tolerant Metacomputing and Generic Name Services : a case study

University of Antwerp, Belgium /
Emory University, USA

- Introduction
- Flashback
- New Research

- => Introduction
- Flashback
- New Research

- => Introduction
  - MPI and Fault-Tolerance
  - This Paper
  - FT-MPI
- Flashback
- New Research

- Introduction
  - => MPI and Fault-Tolerance
  - This Paper
  - FT-MPI
- FlashBack
- New Research

MPI and fault-tolerance (1)

"The MPI standard says little about what happens when something goes wrong. In large part, this is because standards rarely specify the behaviour of erroneous programs or the consequences of events beyond the standard."

(Using MPI 2nd Edition, p. 265)

MPI and fault-tolerance (2)

- “vanilla” MPI :
  - Default error handler : MPI_ERRORS_ARE_FATAL
  - Feasible (?) for stable “big-iron” architectures
  - Less feasible for loosely coupled environment
  - Problematic for environments spanning multiple administrative domains, multiple networks, heterogenous and geographically distributed resources...

MPI and fault-tolerance (3)

- Standard approach : checkpoint-restart (+ message logging)
  - Advantages :
    - works for every problem
    - does not require a rewrite of original MPI-enabled code
    - does not require application to care about fault-tolerance
  - Disadvantages :
    - Heavyweight, potential scalability problems
    - Depending on the nature of the parallel program, less complex solutions might be applicable

      => adapt complexity of recovery to the needs of the software that needs to recover

- Introduction
  - MPI and Fault-Tolerance
  - => This Paper
  - FT-MPI
- Flashback
- New Research

This paper

- Follow-up on research started in 2005 between the university of Antwerp and Emory University
- Context : build an MPI-based computing platform
  - consisting of heterogenous, geographically distributed resources
  - using Emory's H2O metacomputing framework
    - unified remote management
    - easy crossing of multiple administrative domains
  - using UTK's FT-MPI instead of traditional-approach fault-tolerant MPI's for computation

- Introduction
  - MPI and Fault-Tolerance
  - This Paper
  - =>FT-MPI
- FlashBack
- New Research

- Introduction
  - MPI and Fault-Tolerance
  - This Paper
  - => FT-MPI
    - Introduction
    - MPI-level revovery
    - Application-level recovery
    - Advantages / disadvantages
    - FT-MPI basics
- Flashback
- New Research

- Introduction
  - MPI and Fault-Tolerance
  - This Paper
  - FT-MPI
    - => Introduction
    - MPI-level revovery
    - Application-level recovery
    - Advantages / disadvantages
    - FT-MPI basics
- Flashback
- New Research

FT-MPI : Introduction

- Split fault-recovery into
  - MPI-level recovery
  - Application-level recovery

- Introduction
  - MPI and Fault-Tolerance
  - This Paper
  - FT-MPI
    - Introduction
    - => MPI-level revovery
    - Application-level recovery
    - Advantages / disadvantages
    - FT-MPI basics
- Flashback
- New Research

FT-MPI : MPI-level recovery

- Baseline : minimum needed in recovery process of any MPI-based parallel program
- Only restores the job state back to a working condition
- Reconstructs MPI_COMM_WORLD
- Done automatically by FT-MPI itself (configurable)
- Message recovery / consistency :
  - Available for both p2p and collective operations
  - configurable

- Introduction
  - MPI and Fault-Tolerance
  - This Paper
  - FT-MPI
    - Introduction
    - MPI-level revovery
    - => Application-level recovery
    - Advantages / disadvantages
    - FT-MPI basics
- Flashback
- New Research

FT-MPI : application-level recovery

- Application-level recovery is left to the application developer
- All structures containing non-local data must be rebuilt by the application except MPI_COMM_WORLD (multiple configurable modes)

- Introduction
  - MPI and Fault-Tolerance
  - This Paper
  - FT-MPI
    - Introduction
    - MPI-level revovery
    - Application-level recovery
    - => Advantages / disadvantages
    - FT-MPI basics
- Flashback
- New Research

FT-MPI : advantages / disadvantages

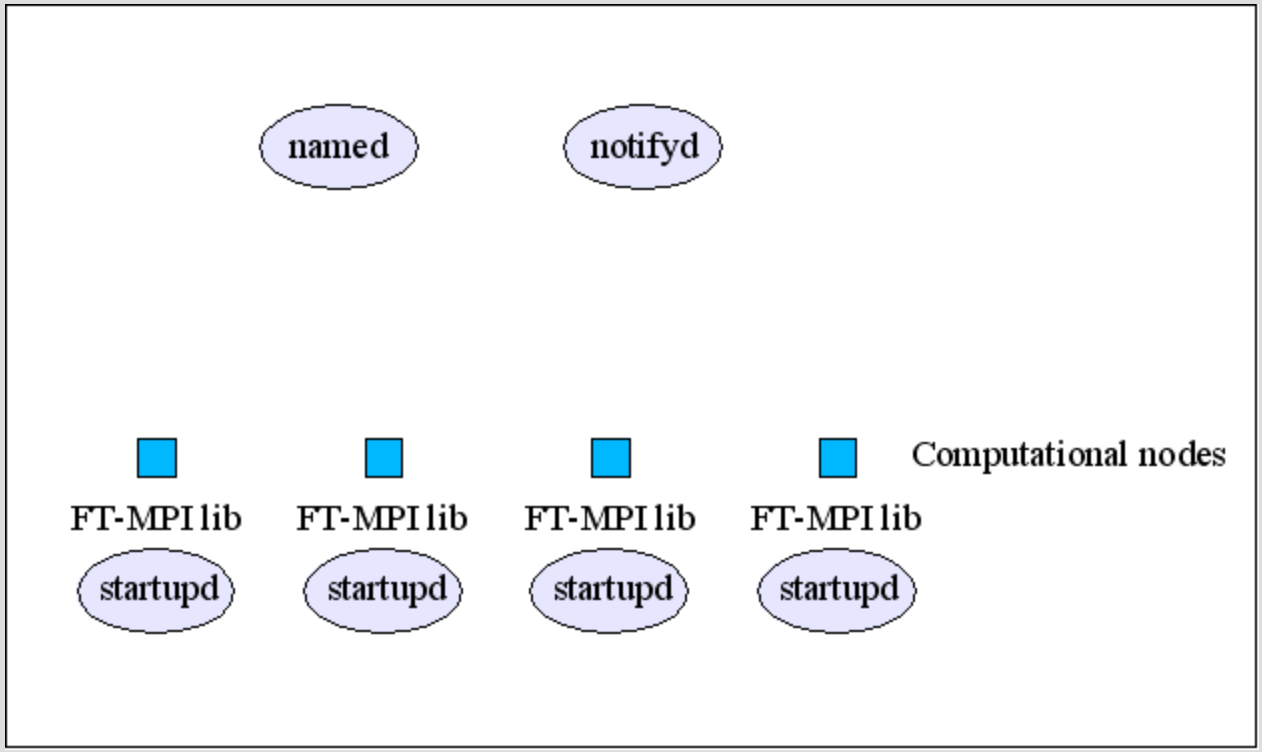
- Advantages :
  - *Can* do own fault-recovery
  - Model scales well to geographically distributed, heterogenous resources
- Disadvantages :
  - *Must* do own fault-recovery
    - might not be trivial for complex parallel algorithms
    - does not come natural to many MPI-programmers

- Introduction
  - MPI and Fault-Tolerance
  - This Paper
  - FT-MPI
    - Introduction
    - MPI-level revovery
    - Application-level recovery
    - Advantages / disadvantages
    - => FT-MPI basics
- Flashback
- New Research

FT-MPI basics (1)

- Design :
  - FT-MPI runtime : message passing, recovery, notification – critical per node
  - startup daemon : process startup, notification – “semi-critical”
  - notify daemon : notification aggregation, non critical.
  - name daemon : keeps information on job state, critical!
- Single Administrative Domain :
  - unique IP address for each participant required
  - each participant must be able to communicate with all others

FT-MPI basics (2)



- Introduction
- => Flashback
- New Research

- Introduction
- => Flashback
  - The name daemon
  - The idea
  - Approach
  - Issues
- New Research

- Introduction
- Flashback
  - => The name daemon
  - The idea
  - Approach
  - Issues
- New Research

The name daemon

- Keeps job state (job members, member coordinates)
  - Essential for recovery of MPI_COMM_WORLD
  - Used at startup and fault-recovery
  - All job state updates done by single “leader” node per job to maintain consistency
  - Updates to job state are propagated to “peons” through registered callbacks
- Issues :
  - Highly state retaining and critical
  - very limited fault-tolerance
  - potential SPoF!

- Introduction
- Flashback
  - The name daemon
  - => The idea
  - Approach
  - Issues
- New Research

The idea

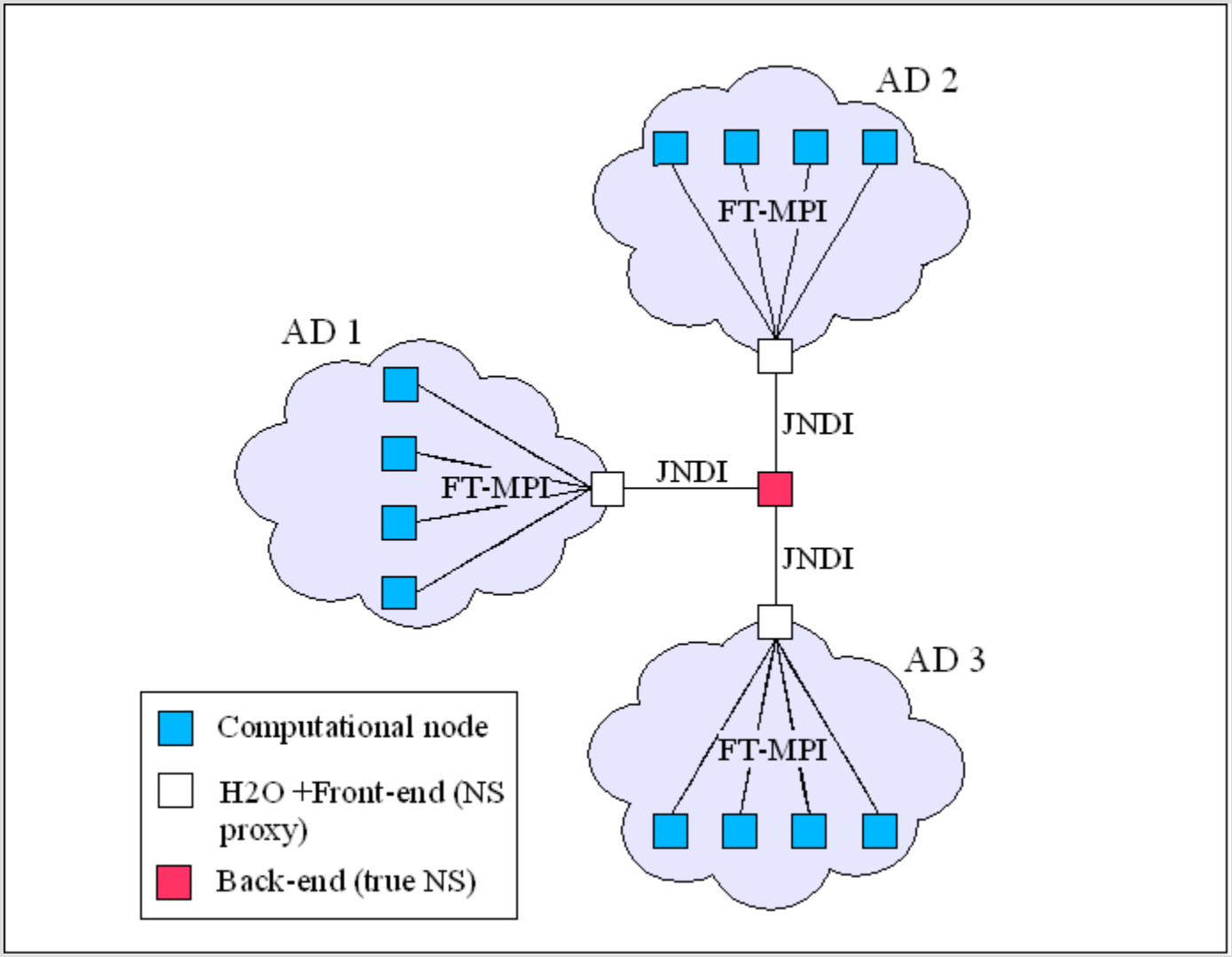
- Current NS does not support features like redundancy, distribution, load balancing, replication or other fault-tolerance or performance related features
- A lot of “off-the shelf” systems do feature such capabilities (both commercial and research projects)
- Use one of the latter instead of the former
- “Plug-a-NS”

- Introduction
- Flashback
  - The name daemon
  - The idea
  - => Approach
  - Issues
- New Research

Approach(1)

- Problem 1 : FT-MPI NS uses a unique protocol
  - Solution : use proxy server to translate calls from original protocol to generic back-end
- Problem 2 : Crossing and managing multiple ADs
  - Solution : use H2O metacomputing framework
    - proxy is a remotely installable “pluglet”
    - unified security and access scheme, centralized management
    - H2O kernel and proxy reside on front-end of cluster
- Problem 3 : generic access to back-end NSs
  - Solution : write proxy in Java

Approach (2)


AD 2
FT-MPI
AD 1
FT-MPI
JNDI
JNDI
JNDI
AD 3
FT-MPI
Computational node
H2O +Front-end (NS proxy)
Back-end (true NS)

Approach (3)

- The good :
  - Cross multiple ADs
  - Less direct connections to NS over cross-AD connections
  - Potential for local optimizations (caching, local resolution of leader election, “multicast” callbacks, ...)
- The bad :
  - Extra level of indirection – performance loss
  - Luckily, the NS is only used at startup and fault-recovery

- Introduction
- Flashback
  - The name daemon
  - The idea
  - Approach
  - => Mapping the NS
- New research

Mapping the NS

- Mapping :
  - FT-MPI NS
    - High-level calls
    - e.g. add record and increment multiple counters in one call
  - JNDI :
    - Low-level calls
  - Compound calls needed!
- Problem 4 : atomicity and concurrency
  - Solution : use remote atomic locking scheme to limit concurrent access to data

- Introduction
- Flashback
- => New research

- Introduction
- Using a generic NS
- => New research
  - Rationale
  - Approach
  - Minimize locking during recovery
  - Eliminate locking during election
  - Experimental results

- Introduction
- Using a generic NS
- New research
  - => Rationale
  - Approach
  - Minimize locking during recovery
  - Eliminate locking during election
  - Experimental results

Rationale

- The previously presented solution :
  - uses locking for all operations on resources which might be accessed by multiple proxies at once
  - advantage :
    - shared resource protection using locks is *generic* (not just usefull for FT-MPI)
  - disadvantage :
    - it is also expensive...
      - each compound operation requires lock + unlock
        = at least 2 extra interactions with back-end
      - extra checks required when trying to acquire an already locked resource
        = periodic extra interactions

- Introduction
- Using a generic NS
- New research
  - Rationale
  - => Approach
  - Minimize locking during recovery
  - Eliminate locking during election
  - Experimental results

Approach

- Problem 1 : locking is expensive
  - Solution : minimize the use of locks
- The majority of calls to the NS involve
  - Fault recovery
  - Leader election
- Minimizing or eliminating locking during these two phases will drastically reduce the amount of locks needed, and improve performance

- Introduction
- Using a generic NS
- New research
  - Rationale
  - Approach
  - => Minimize locking during recovery
  - Eliminate locking during election
  - Experimental results

Minimize locking during recovery (1)

- Problem 2 : minimize locks during recovery :
  - Solution :
    - all updates to job state are done by one node : the leader
    - these updates do not require locks
- Number of locks during job recovery drastically reduced
- But operations remain compound
  - Leader might fail in mid-update
  - Leaves back-end in non-consistent state

Minimize locking during recovery (2)

- Problem 3 : mid-compound operation failures
  - Solution :
    - all original data is left intact until full completion of a compound operation
    - this data is pointed to by a single, ultimately authorative “index record”
    - as long as this index record is not overwritten, all original data remains authorative as well
    - all new and changed data is written to temporary records
    - when all updates have been done, overwrite index record with up-to-date indices
    - the new index record, with all indicated compound data becomes authorative with a single atomic bind operation

- Introduction
- Using a generic NS
- New research
  - Rationale
  - Approach
  - Minimize locking during recovery
  - => Eliminate locking during election
  - Experimental results

Eliminate locking during election (1)

- Problem 4 : eliminate locking during leader election
  - Solution : an adapted election algorithm
- The new algorithm...
  - Expands upon and modifies basic methods for locking
  - uses the equivalence of “Getting a lock” to “winning election”
  - Needs no record update or unlock beyond the first interaction
  - Does not use leases
  - Uses local optimization for greater performance gain
  - Still happens fully transparently to nodes

Eliminate locking during election (2)

- The Algorithm :
  - NS keeps a token symbolizing leadership
  - NS also keeps counter for no. of elections
  - Counter is read by proxy at startup and cached
  - During an election :
    - proxies : cached counter ++, once
    - resolve election locally for each node sharing the same proxy
    - each proxy tries to bind <*leadertoken*>_<*electioncounter*>
    - the successful proxy rebinds the leader token
    - callbacks are used to warn all other proxies of the outcome, and all appropriate results are relayed to the individual nodes

- Introduction
- Using a generic NS
- New research
  - Rationale
  - Approach
  - Minimize locking during recovery
  - Eliminate locking during election
  - => Experimental results

Experimental Results (1)

- Test focuses on intended application :
  - geographically distributed, heterogeneous resources
- Setup :
  - preliminary tests : proof-of-concept implementation
  - two machines
  - separated by internet connection
  - one machine in Atlanta, Georgia – the other in Antwerp, Belgium

Experimental Results (2)

- Backends :
  - OpenLDAP with Berkeley DB backend
    - standard LDAP-based directory server
    - light on lookup
    - heavy on insert
  - HDNS
    - Harness Distributed Name Service
    - Originally created for Harness Metacomputing Framework
    - Ported to Java and updated for H2O Metacomputing Framework
    - Uses its own protocol, but supports JNDI SPI
- Compare new setup with original NS

Experimental Results (3)

- Test :
  - feasability
  - scalability
  - stability
- Experiment 1 :
  - for insert and read
  - test scalability as payload per entry rises (transaction size)
  - measure wallclock time
- Experiment 2 :
  - for insert and read
  - test scalability as the amount of entries rises

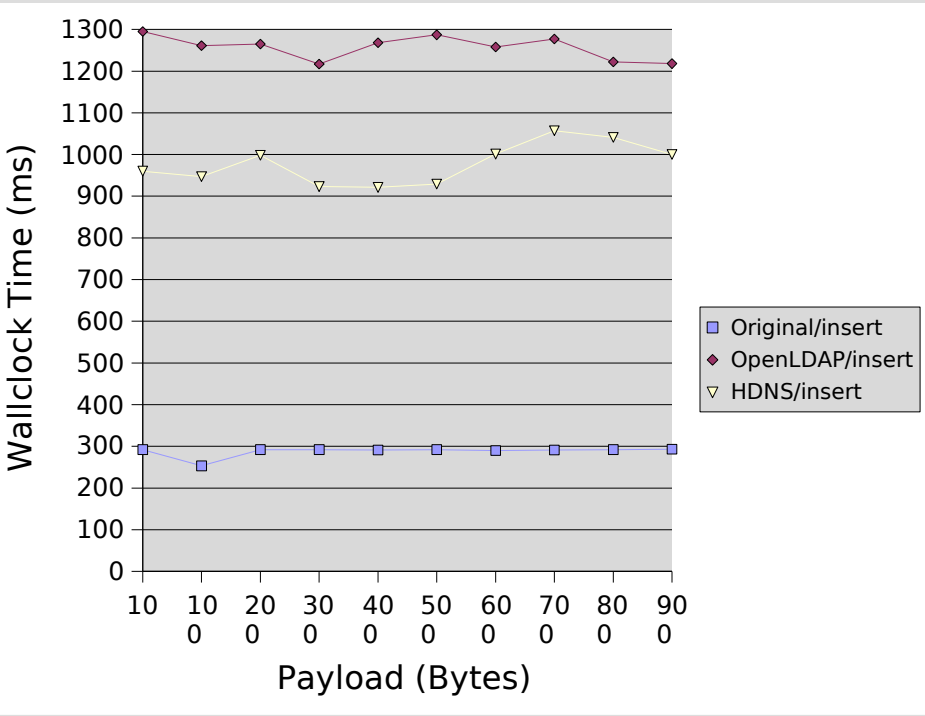
Experimental Results (4)

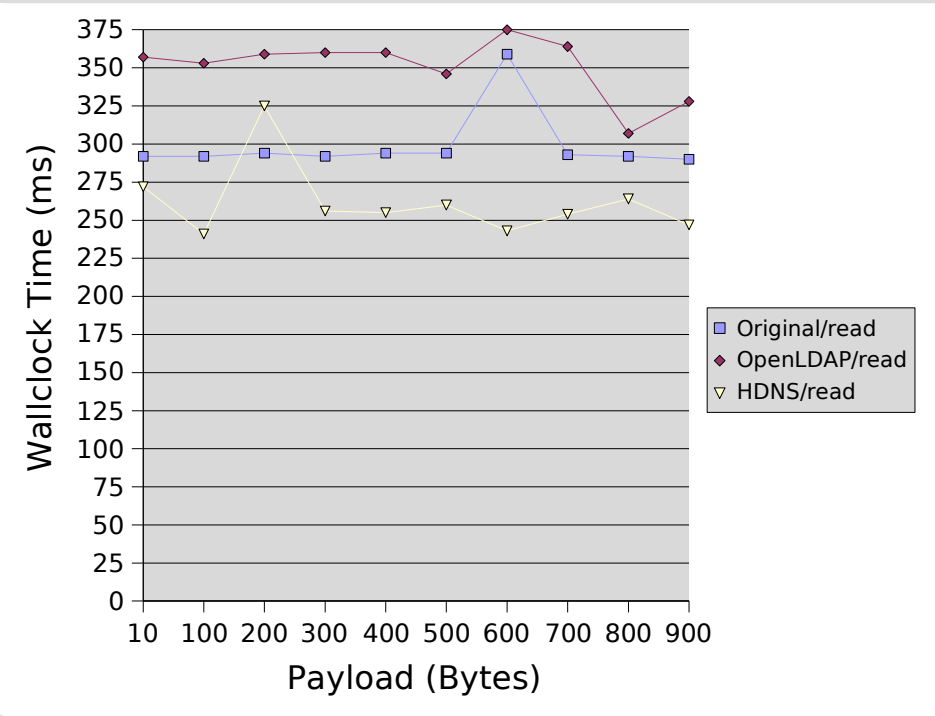
- results :
  - transparent switching between backends works as advertised
  - HDNS performs better than OpenLDAP for both experiments
    - as expected, OpenLDAP performs worse on insert
    - on read, OpenLDAP performs about as well as original NS...
    - ...but even here, HDNS manages to outperform it
      - OpenLDAP (compiled code)
        - has advanced caching and other features
        - but these are not used in this particular application
      - HDNS (Java!)
        - unsophisticated (relatively to OpenLDAP)

Experimental Results (5)

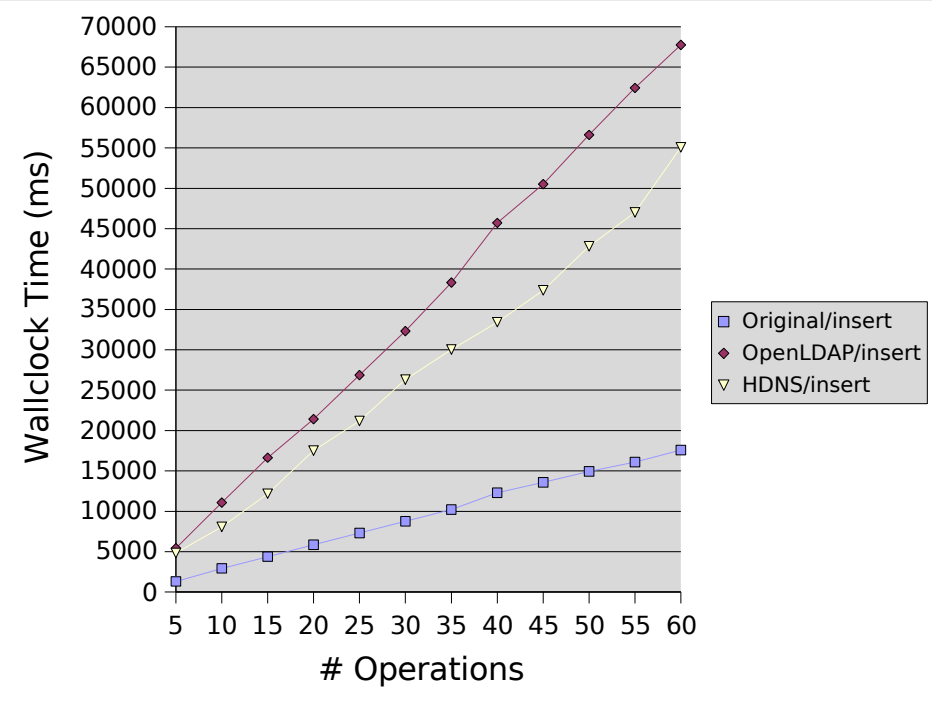
- further results :
  - OpenLDAP performs *a lot* better with the reduced amount of locks
    - still slower than original NS
    - but more than twice as fast as in original experiments
    - HDNS performance behaves similarly
    - further optimizations using the local proxies (like the local resolution of leader election) should provide further performance gain
  - measured wallclock times still linear
    - system is still scalable
    - delays are still predictable

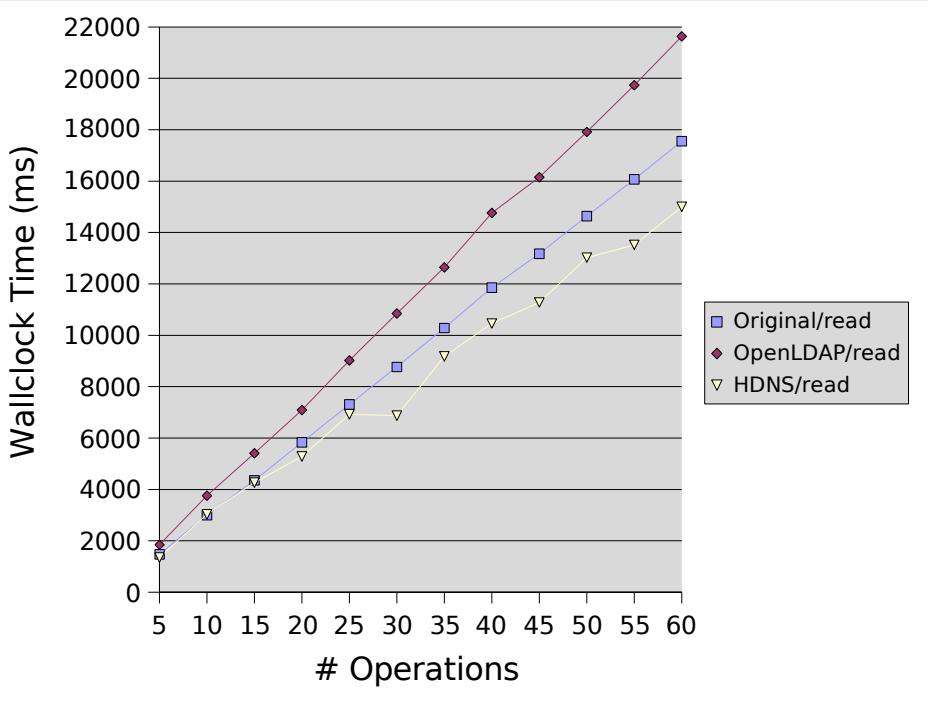
Experimental Results (6)





Experimental Results (7)





- Questions?