

Holistic Fault Tolerance for Petascale Computing

Al Geist
Oak Ridge National Laboratory
Oak Ridge, TN U.S.A

Dagstuhl Germany
Seminar 06071
February 12-17, 2006

Research sponsored by U.S. Department of Energy

What is the fastest way to slow down A Petascale Computer?

- A. Run Parallel Matlab
- B. Run an I/O intensive job
- C. Have a system failure
- D. All of the above

This presentation will discuss petascale issues for a particular I/O intensive job called checkpoint/restart
And discuss system failures, detection, and recovery

System Fault Recovery – A story of stone tools and bear skins

A user notices that a job has not completed in a normal way, and contacts the system administrator.

The sysadmin tries to log onto the node, which may or may not succeed.

In many cases, even if the login succeeds, the sysadmin can not find a cause, and reboots the node as a prophylactic measure

If the login fails, the sysadmin pings the node. Even if the ping works, there is no way to get onto the node, so the sysadmin reboots the node

If the ping fails, then the only option is to reboot the node

I/O for Petascale

Assume a “balanced” 1 PF machine with 1 TB/s I/O

500 controllers and thousands of disks

Could cost \$30M-\$60M in 2008

A 1 TB/s filesystem in 2008 requires approximately
2,500 file servers

One choice is to reduce the system I/O bandwidth

But this increases the time to checkpoint applications

Fault Tolerance – a petascale perspective

Petascale systems will have 100,000+ processors on multi-core chips.

The time before **some** failure will be measured in minutes.

Checkpointing and restarting this large a system could take longer than the time to the next failure!

What to do? Autonomic? Self-healing?
Adaptable? Malleable?

Development of algorithms that can be **naturally fault tolerant** I.e. failure anywhere can be ignored? And still get the right answer.

- No monitoring
- No notification
- No recovery

We have created examples...

Is it possible?



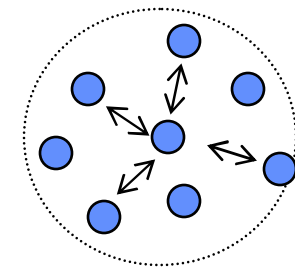
Progress Natural Fault Tolerant algorithms



Demonstrated that the scale invariance and natural fault tolerance can exist for local and global algorithms where 100 failures across 100,000 processes

Finite Difference (Christian Engelman)

- Demonstrated natural fault tolerance w/ chaotic relaxation, meshless, finite difference solution of Laplace and Poisson problems



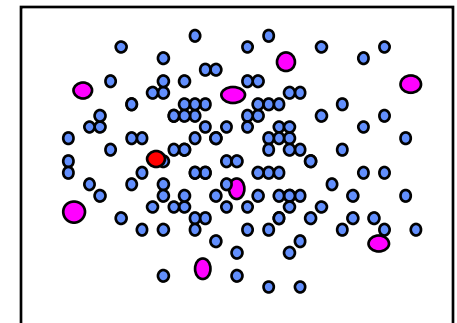
local

Global information (Kasidit Chancio)

- Demonstrated natural fault tolerance in global max problem w/random, directed graphs

Gridless Multigrid (Ryan Adams)

- Combines the fast convergence of multigrid with the natural fault tolerance property. Hierarchical implementation of finite difference above.
- Three different asynchronous updates explored



global

The System Can't Ignore Faults

Holistic Fault Tolerance

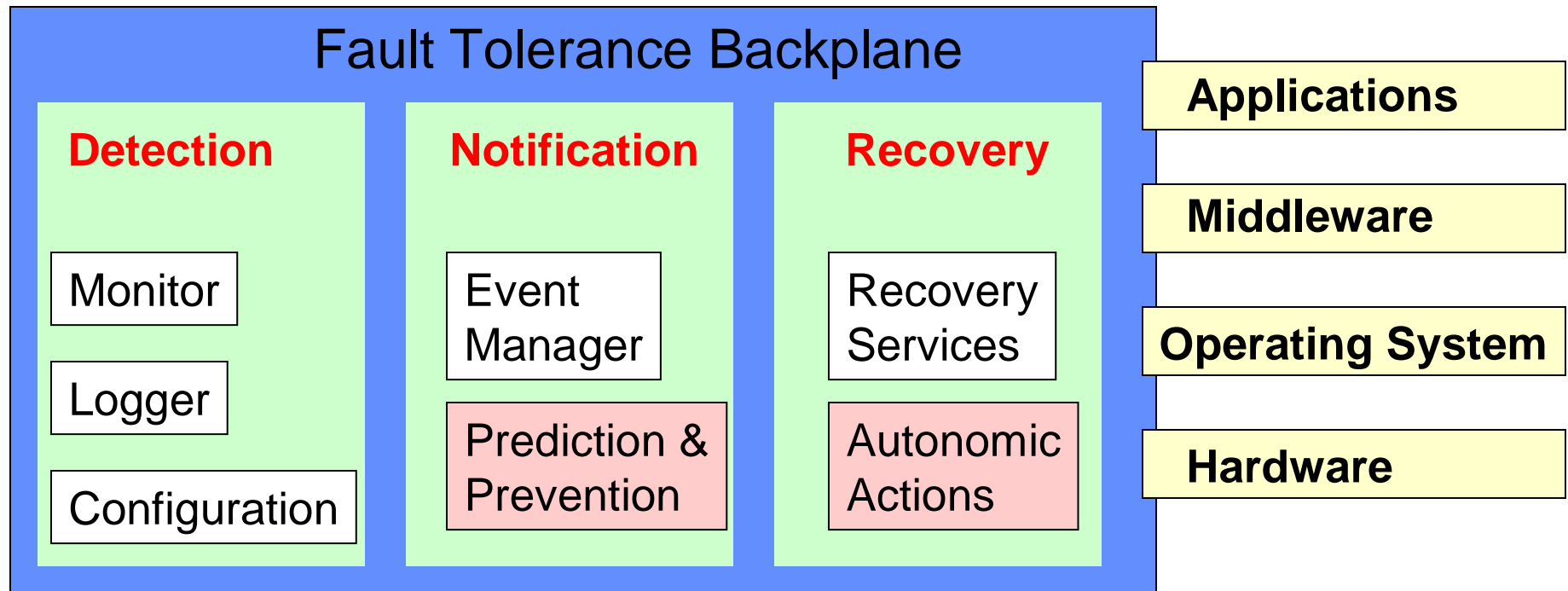
- Research into active fault management that takes into account the full impact of faults across application, middleware, OS, and hardware (heterogeneous nodes – compute, I/O, service)
- Complement and extend RAS systems

Fault tolerance in systems software – the best type of fault is the one that never happens

- Research into prediction and prevention
- Survivability and resiliency when faults can not be avoided i.e. unpredictable events

Holistic Fault Tolerance

Vertical Backplane to provide Fault awareness, prediction and recovery across the entire HPC system from the application to the hardware



Prediction

When and Where inside the system is failure imminent?
Statistical analysis of real-time monitoring coupled with historical logs of past failures.

Data from:

System monitors eg. Temperature, fan speed, traffic

Middleware monitors eg. High I/O loads on service node X

Application history eg. AI's code always hangs the system



Autonomic Recovery

Application Recovery Options

Restart – from checkpoint or from beginning

Notify application and let it handle the problem

Migration of task to other hardware

Reassignment of work to remaining tasks

Replication of tasks across machine

Ignore the fault altogether



Final Thought – How can you be sure?

Validation of answer on such large systems

Fault may not be detected

Recovery introduces perturbations

Result may depend on which nodes fail

Result looks reasonable but is actually wrong

Can't afford to run every job three times

I'll just keep running
the job till I get the
answer I want



HW errors SNL experience
Act of God UVa
Numerical accuracy 64 loss
Load balance