

Fault-Tolerance Challenges and Solutions



Presented by

AI Geist

Computer Science Research Group
Computer Science and Mathematics Division

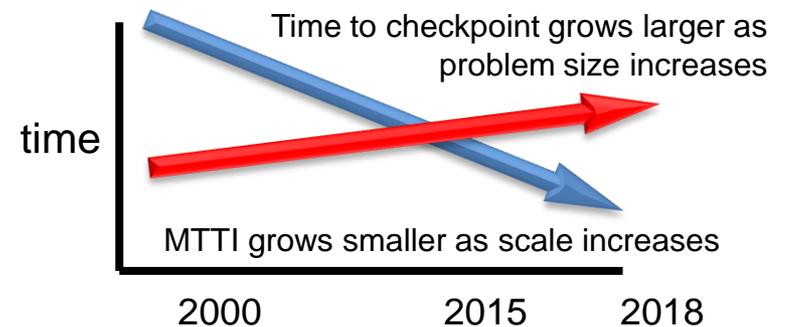
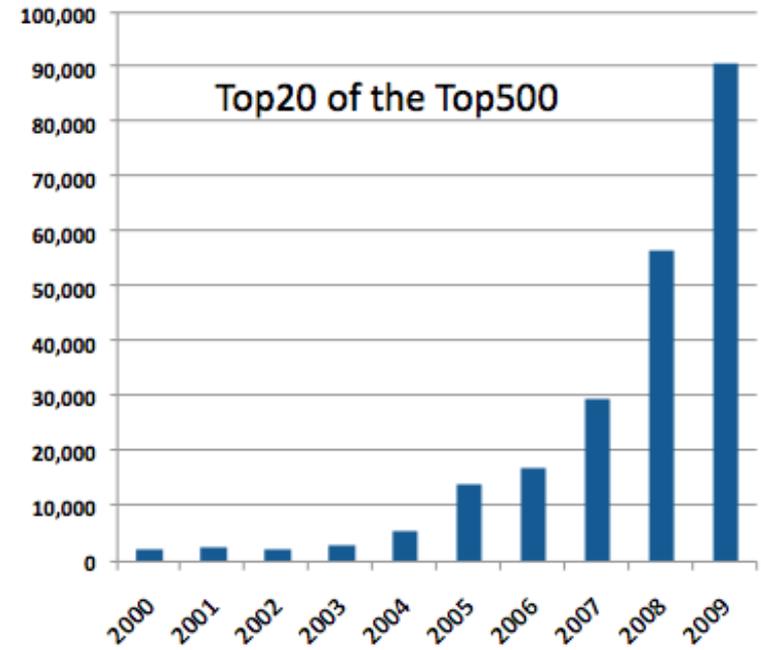


Rapid growth in scale drives fault tolerance need

Challenges

- Fundamental assumptions of applications and system software design did not anticipate exponential growth in parallelism
- Number of system components increasing faster than component reliability
- Mean time between failures of minutes or seconds for exascale built today
- Silent error rates increasing
- Checkpoint/restart overhead too large for future systems

Average Number of Processors Per Supercomputer



Hardware reasons for the worsening problem

1. **Number of components** (both memory and processors) will increase by an order of magnitude, which will increase hard and soft errors
2. **Smaller circuit sizes, running at lower** voltages to reduce power consumption, increases the probability of switches flipping spontaneously due to thermal and voltage variations as well as radiation, increasing soft errors
3. **Power management cycling** significantly decreases the components' lifetimes due to thermal and mechanical stresses
4. **Resistance to add additional HW** detection and recovery logic right on the chips to detect silent errors, because it will increase power consumption by 15% and increase the chip costs
5. **Heterogeneous systems** make error detection and recovery even harder; for example, detecting and recovering from an error in a GPU can involve hundreds of threads simultaneously on the GPU and hundreds of cycles in drain pipelines to begin recovery

Software reasons for the worsening problem

- 1. Existing fault tolerance techniques** (global checkpoint/global restart) **will be unpractical at exascale**
- 2. There is no standard fault model.** Neither is there a standard fault test suite nor metrics to stress resilience solutions and compare them fairly
- 3. Errors, fault root causes, and propagation are not well understood**
- 4. MPI does not offer a paradigm for resilient programming.** A failure of a single task often leads to the killing of the entire application
- 5. Present applications and system software are neither fault tolerant nor fault aware** and are not designed to confine errors/faults, to avoid or limit their propagation, and to recover from them when possible

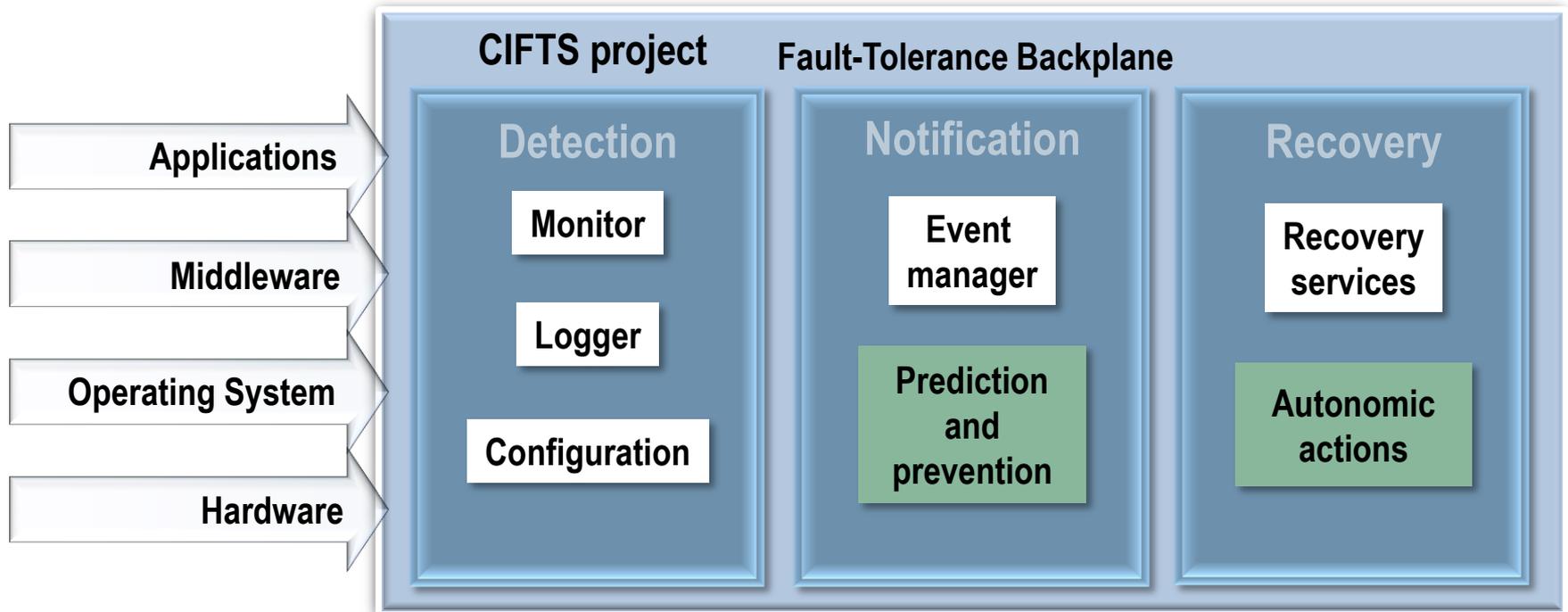
ORNL is leading research to solve the Exascale Resilience Challenges

Research Areas

- **Local recovery and migration**
- **Development of a standard fault model and better understanding of types/rates of faults**
- **Improved hardware and software reliability**
- **Greater integration across entire stack**
- **Fault-resilient algorithms and applications**

Need a standard fault model and a holistic solution

We need coordinated fault awareness, prediction, and recovery across the entire HPC system from the application to the hardware



“Prediction and prevention are critical because the best fault is the one that never happens”

Project under way at ANL, ORNL, LBL, UTK, IU, OSU

Fault model:

Three steps and error types

1. Detection that something has gone wrong

- **System:** Detection in hardware
- **Framework:** Detection by runtime environment
- **Library:** Detection in math or communication library

2. Notification of the application, runtime, or system components

- **Interrupt:** Signal sent to job or system component
- **Error code returned by application routine**

3. Recovery of the application to the fault

- **By the system**
- **By the application**
- **Neither: Natural fault tolerance**

Types of errors (h/w or s/w)

- **Hard errors:** permanent errors that cause system to hang or crash
- **Soft errors:** transient errors, either correctable or short term failure
- **Silent errors:** undetected errors either permanent or transient. *Concern is that simulation data or calculation has been corrupted and no error reported*

Fault model: Six options for system to handle failures

- **Restart—from checkpoint or from beginning**
- **Notify application and let it handle the problem**
- **Migrate task to other hardware before failure**
- **Reassign work to spare processor(s)**
- **Replicate tasks across machine**
- **Ignore the fault altogether**

**Need standard API for each application (or component)
to specify to the system what to do if a fault occurs**

Fault model: Five recovery modes for MPI applications

Harness project's FT-MPI explored five modes of recovery

- **ABORT:** Just do as vendor implementations
- **BLANK:** Leave holes (but make sure collectives do the right thing afterward)
- **SHRINK:** Reorder processes to make a contiguous communicator (some ranks change)
- **REBUILD:** Respawn lost processes and add them to `MPI_COMM_WORLD`
- **REBUILD_ALL:** Same as `REBUILD` except that it rebuilds all communicators, and groups and resets all key values, etc.

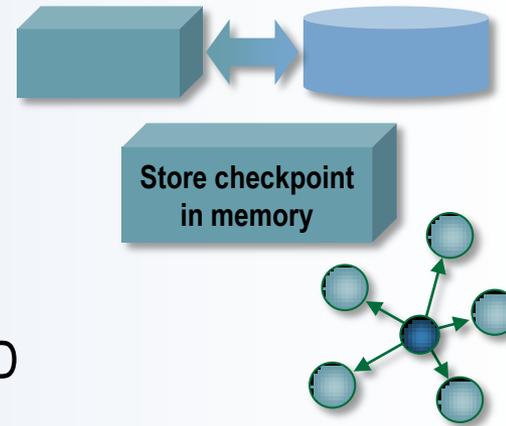
These modes affect the size (extent) and ordering of the communicators

The MPI-3 forum is discussing these and other options that allow applications to recover from faults

Need to develop new paradigms for applications to handle faults

Some state saved

1. Restart from checkpoint file
[large apps today]
2. Restart from diskless checkpoint
[avoids stressing the I/O system and causing more faults]
3. Recalculate lost data from in-memory RAID



No state saved

4. Lossy recalculation of lost data
[for iterative methods]
5. Recalculate lost data from initial and remaining data
6. Replicate computation across system
7. Reassign lost work to another resource
8. Use natural fault-tolerant algorithms

Need to develop rich methodology to “run through” faults

Increasing need for detection and validation

Validation of an answer on such large systems is a growing problem; simulations are more complex; solutions are being sought in regions never before explored

- Fault may not be detected
- Recovery introduces perturbations
- Result may depend on which nodes fail
- Result looks reasonable, but it is actually wrong

- Can't afford to run every job three (or more) times
- Yearly allocations are like \$5M–\$10M grants

Contact

AI Geist

Computer Science Research Group
Computer Science and Mathematics Division
(865) 574-3153
gst@ornl.gov