Argonne NATIONAL LABORATORY

Application-driven Fault-Tolerance for High Performance Distributed Computing.

Bogdan Nicolae
Argonne/MCS

George Bosilca
University of Tennessee at Knoxville

Agenda

- **Introduction: 0h30**
  - 5 minutes motivation (failure rates) - Bogdan
  - 10 minutes checkpoint/restart (application level versus system level) - Bogdan
  - 15 minutes alternatives (ULFM, Replication, etc.) – George
- **VeloC and Hands-on installation: 1h30 – Bogdan**
  - 25 mins -> VeloC presentation
  - 5 mins -> Questions
  - 45 mins -> Hands-on-session
    - 15 minutes of setup (Docker, example source code)
    - 30 minutes of VeloC
      - Configuration: 10 minutes
      - Filling the gap in the heat equation application: 10 minutes
      - Playing with failures: 10 minutes
- **ULFM: 1h30 – George**
  - 25 mins -> ULFM presentation
  - 5 mins -> Questions
  - 45 mins -> Hands-on-session
    - 30 min on ULFM
    - 15 minutes on ULFM + VeloC

Why FT for HPC?

- FT need for HPC was marginal because HPC system MTBF were high enough (1 week, 1 month). This is not true anymore for large systems today (MTBF of 1 day and less are seen)
- It can only get worse with the increase of the number of components and component complexity
- There is no compromise:
  - *Fault tolerance is not like other problems of HPC (performance, efficiency, power consumption, etc.)*
  - *There is no half success: application execution succeeds with correct results or fails!*
- Clouds are starting considering HPC applications and Cloud nodes have typically a much lower MTBF than HPC nodes.

Some important definitions

From Avizienis, Laprie et al.:

Definition from the notion of service: a sequence of the system's external states *(perceived by users)*

Correct service: is delivered when the service implements the system function.

Service failure: is an event that occurs when the delivered service deviates from correct service.

- Failure: at least one (or more) external state of the system deviates from the correct service state (*ex: a computing node fails, a parallel execution fails, the application fails)*

- Error: part of the *internal system state* that may lead to … service failure

- Fault: The adjudged or hypothesized cause of an error *(root cause of the failure)*

Some important definitions

From Avizienis, Laprie et al.:

Definition from the notion of service: a sequence of the system's external states *(perceived by users)*

Co

Se

Example:

A) A particle hits a DRAM cell and generates a fault
B) The fault changes the DRAM cell state and becomes an error
C) The error does not affect the rest of the system until a process reads the cell
D) The error propagates as a failure if after the read of the memory cell the software computation, control or I/O deviates from the behavior it would have had from a correct memory cell

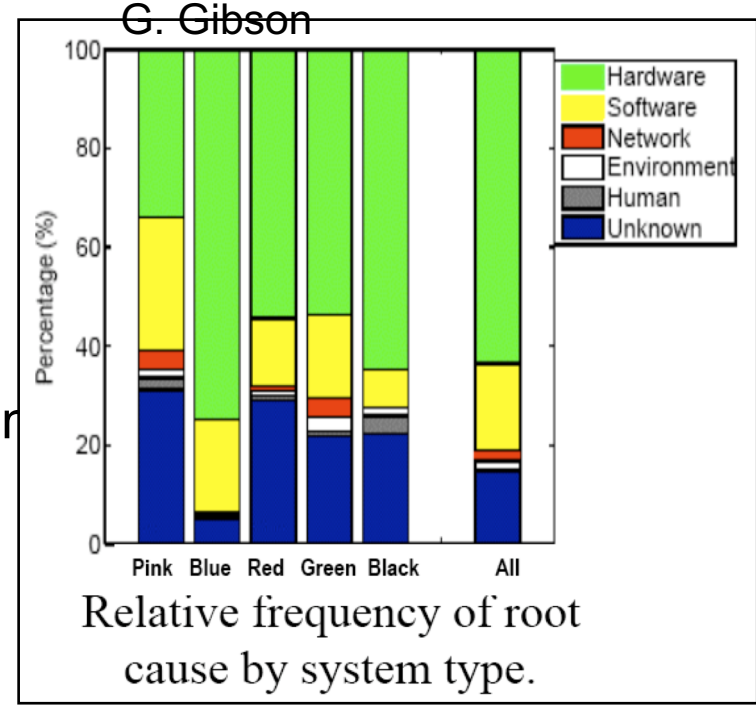
- Fault: The adjudged or hypothesized cause of an error *(root cause of the failure)*

Specific Error Outcomes in HPC

Types of errors:

- Power outage
- Hard errors (broken component: memory, network, core, disk, etc.)
- Detected soft errors (bit flip in memory, logic, bus)
- OS error (buffer overrun, deadlock, etc.)
- System Software error (service malfunction
- Application bugs
- Administrator error (Human)
- User errors (Human)

G. Gibson



Relative frequency of root cause by system type.

Classes of errors:

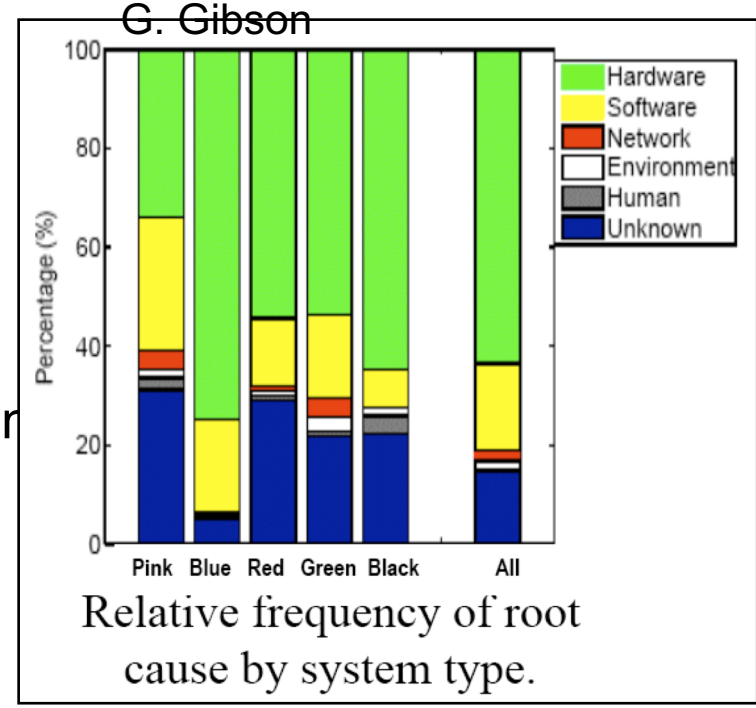
- Detected and corrected (by ECC, Replication, Re-execution)
- Detected and uncorrectable (leading to application crash)
- Undetected (leading to data corruption, application hang, etc.)

Specific Error Outcomes in HPC

Types of errors:

- Power outage
- Hard errors (broken component: memory, network, core, disk, etc.)
- Detected soft errors (bit flip in memory, logic, bus)
- OS error (buffer overrun, deadlock, etc.)
- System Software error (service malfunction
- Application bugs
- Administrator error (Human)
- User errors (Human)

G. Gibson



Relative frequency of root cause by system type.

Classes of errors:

- Detected and corrected (by ECC, Replication, Re-execution)
- Detected and uncorrectable (leading to application crash)
- Undetected (leading to data corruption, application hang, etc.)

MTBF 10 Years Ago?

Blue Gene Hardware Reliability: Argonne Data

Paul Coteus, IBM

- **BG/L System Design Target:**
  - 64 Racks/131k cores MTBF should be greater than 7 Days
- **Comparison of actual data made by ANL Labs**
  - asked a number of facilities for reliability data.
- **Multi teraflop IA64 or X86 systems have 100's to 1000's of individual compute nodes.**
- **For comparison between different systems, fail rates are normalized to peak system performance in teraflops**

System Type	Peak System Performance (Teraflops)	Full System Mean Time Between Failures (Days)	Failures per Month	Failures per Month per Teraflop	Failures per Month per BG Rack
IA64	3.0	1.3	24.0	8.000	
IA64	10.7	1.1	28.3	2.645	
x86	1.7	4.5	6.7	3.941	
x86	17.2	0.7	45.1	2.622	
Power 5	15.0	1.1	19.0	1.267	
Blue Gene	365.0	7.5	4.0	0.011	0.06

MTBF 5 years ago

- Two classes:
  - Based on proprietary components: IBM designs BG line with a full system MTBF of 7 days (true for BG/L, BG/P, BG/Q?)

Component	FIT per Component	Components per 64K System	FIT per System
DRAM	5	608,256	3,041K
Compute + I/O ASIC	20	66,560	1,331K
ETH Complex	160	3,024	484K
Non-redundant power supply	500	384	384K
Link ASIC	25	3,072	77K
Clock chip	6.5	1,200	8K
Total FITs			5,315K

Table 6.12: BlueGene FIT budget.

Paul Coteus, IBM

  - Using commodity components (Intel, AMD processors, etc.): MTBF of about 1 day (some less, some more) for systems with 100,000+ cores

Jaguar XT5 status, April 2009

-

Stable	• MTTI: 32 hours • MTTF: 52 hours	• Driver for downtimes: Spider testing • System has been up as long as 10 days

Current failure rates

Fault	Local Consequence	Cascading Consequence	Mean Time between Faults
Node failures (some hardware or OS part of the node fail leading to a complete failure of the node)[2]	User processes running on the node crashes	Full user execution crash because the runtime of the resource/job manager decides to kill the execution (R1) or because of a cascading to full system outage (R2)	BW[3] 6.7h [Mar14] Titan[3]: 7.5 h [Tiw14] resulting in mean time to application failure of 40 h [TGR15]
Network failure	The user processes that cannot communicate experience time-outs on communication. OS or runtime may kill these processes. The affected processes may crash on their own. However, user processes may be able to tolerate transient network shoot down/rerouting.	Potential full user execution crash because of R1. Also if the execution was not able to checkpoint because of network failure, then it will need to restart from the previous checkpoint (C1).	BW: 20 h (link failure) [Mar14]
File system failure	The user processes that cannot perform file access experience time-outs. OS or runtime may suspend or kill these processes. The affected processes may	Potential full user execution crash because of R1 or R2 or because the execution reached the wall-time limit. C1 applies here as well.	BW: 35.4 h (between execution failures – scratch partition).

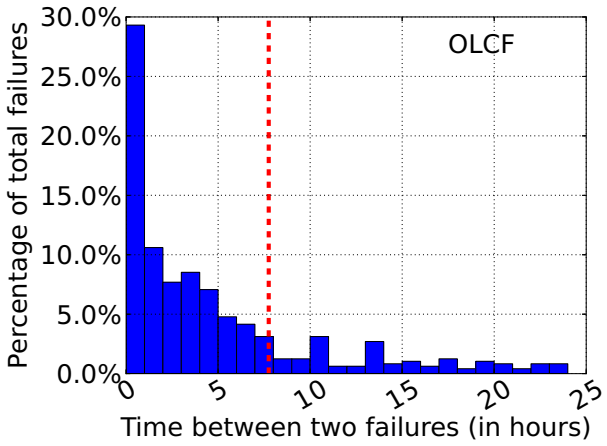
2 For example, GPU bus errors (disconnection of the GPU), voltage fault, kernel panic, PCI width degrade, machine check exception, and SXM (PCI) power off observed in Titan lead to process crashes [Gup15].

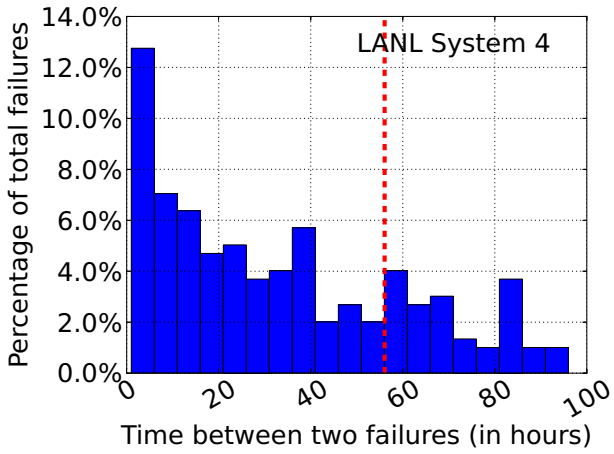
3 Time between failures of any node in the system. Each node MTBF is typically 25 years in these systems [Tiw14].

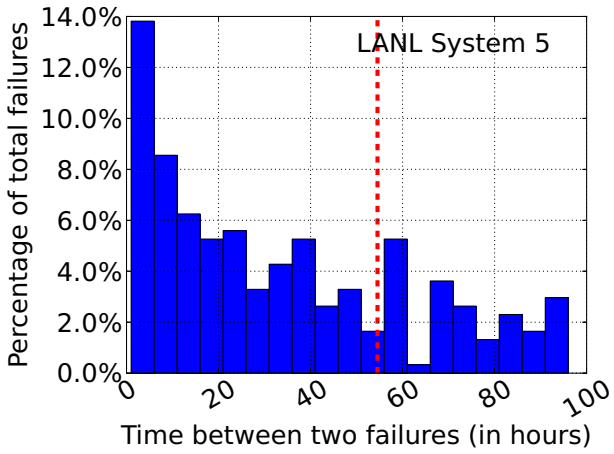
Interval between failure can be << MTBF

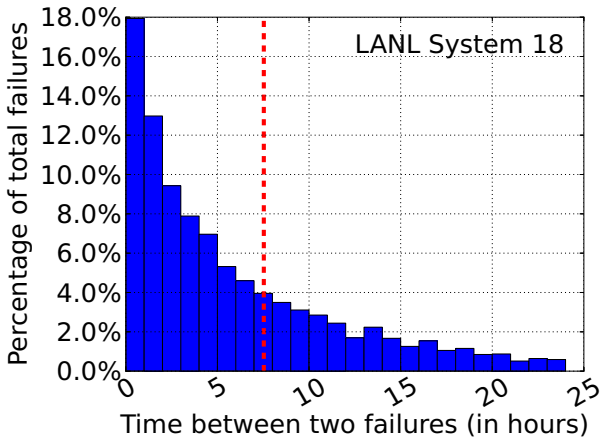
This observation holds for other systems (including old ones)

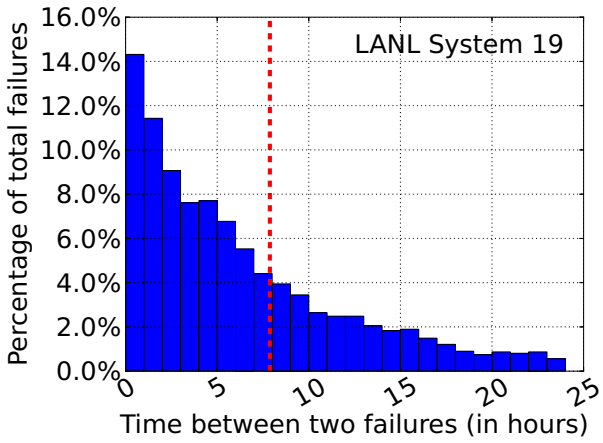
MTBF

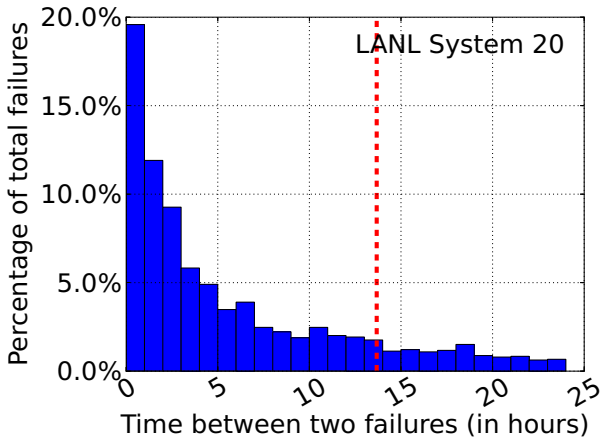






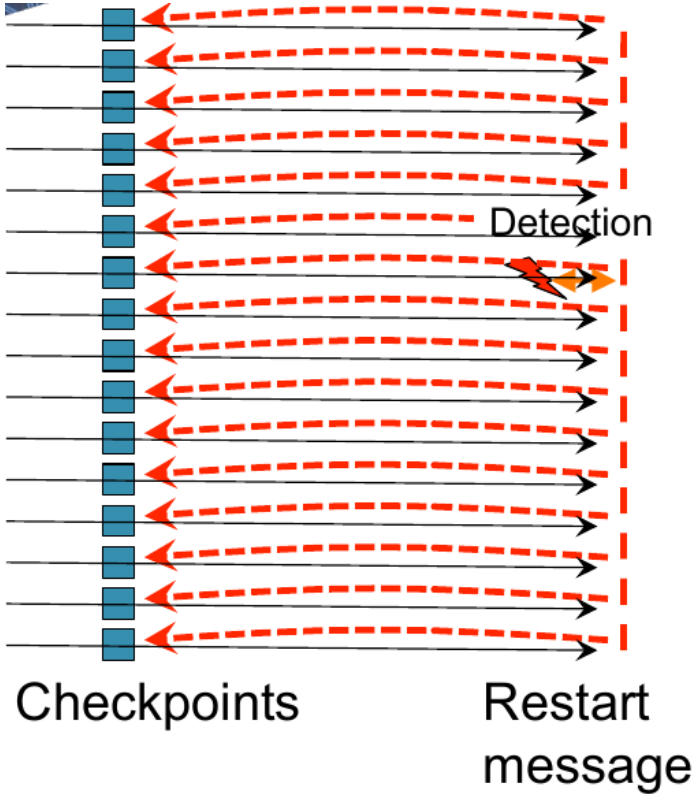






Devesh Tiwari, Saurabh Gupta, Sudharshan Vazhkudai, Lazy Checkpointing: Exploiting Temporal Locality in Failures to Mitigate Checkpointing Overheads on Extreme-Scale Systems, Proceedings of the Annual IEEE/IFIP Int'l Conference on Dependable Systems and Networks (DSN), 2014.

Principle of Checkpoint-Restart



Checkpointing techniques

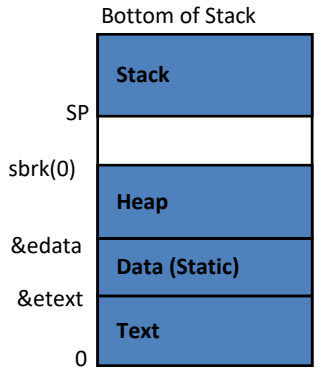
- A checkpoint is just a ‘snapshot’ of a process (or system) at a certain point in time
- A checkpointing system provides a way to take these snapshots, and to restart from them

Type of checkpoint mechanisms:

VM Level
- Completely transparent
- Heavyweight (full state of each process + OS)

Kernel & User (System) Level
- Easy to add checkpointing to existing code
- Works with (almost) any programs
- ‘Coarse’ grain approach (full state of each process)
  - Examples: Libckpt, CRIU, DMTCP

Application Level
- Could require modifications
- Different API (memory level, file level)
- ‘Fine’ grain approach (only critical state of each process)
- Examples: FTI, SCR, VeloC

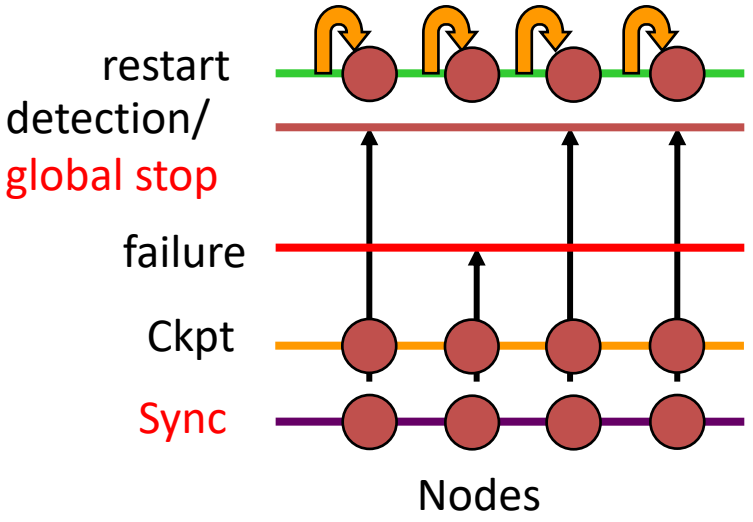


Parallel Checkpointing

Coordinated Checkpoint

The objective is to checkpoint the application when there is no in transit messages between any two nodes
Coordination:

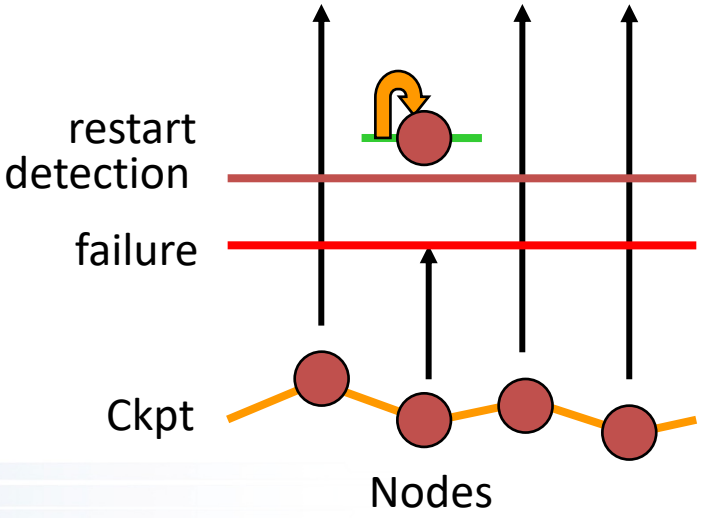
- Automatic through a protocol
- Implicit: application level



Uncoordinated Checkpoint

No need for global coordination (scalable)

- Nodes may checkpoint at any time (independently of the others)
- Need to log undeterministic events, i.e., in-transit messages
- Too complex



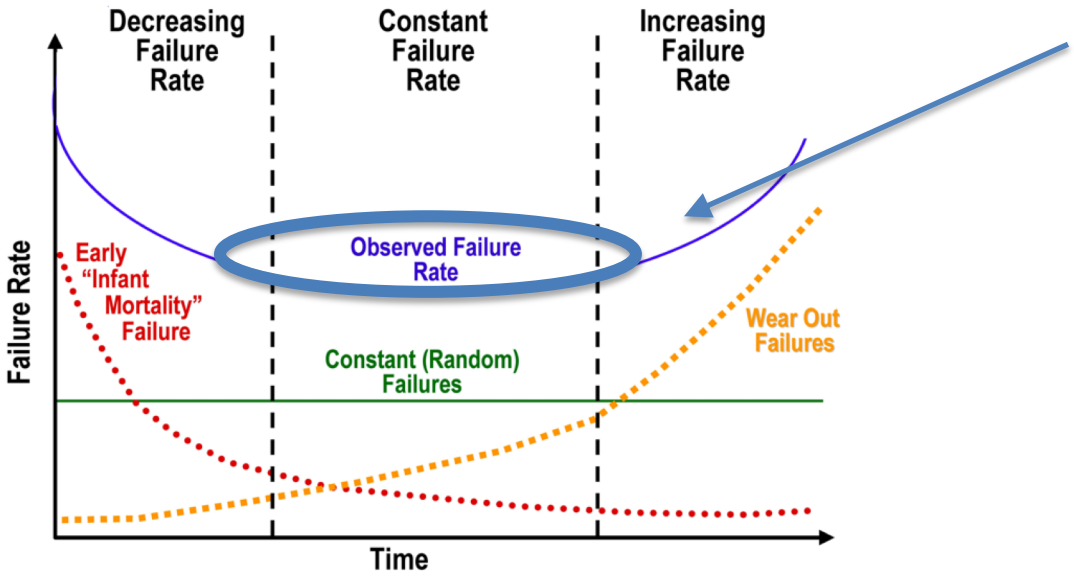
Application Level Implicitly Coordinated Parallel Checkpointing

Example: heat distribution bulk sync code

```
    while(i < ITER_TIMES) {
        localerror = doWork(nbProcs, rank, M, nbLines, g, h);
        if (((i % ITER_OUT) == 0) && (rank == 0))
            printf("Step : %d, error = %f\n", i, globalerror);
        if ((i % REDUCE) == 0)
            MPI_Allreduce(&localer, &globaler, 1, MPI_DOUBLE, MPI_MAX,
MPI_COMM_WORLD);
        if (globalerror < PRECISION)
            break;
        i++;
        if (i % CKPT_FREQ == 0) {
             FILE *outFile = fopen(“checkpoint”, “wb”);
             fwrite(&I, sizeof(int), 1, outFile);
             fwrite(h, sizeof(double), M * nbLines, outFile);
             fwrite(g, sizeof(double), M * nbLines, outFile);
        }
    }
```

When to checkpoint

[Young 74] Let's assume that our system failure rates follow the bath tub pattern.
We are interested to compute the checkpoint interval for the constant failure rate regime



Well modeled by the
Exponential distribution
Failure density function:

$$f(x;\lambda) = \begin{cases} \lambda e^{-\lambda x}, & x \geq 0, \\ 0, & x < 0. \end{cases}$$

MTBF = 1 / λ
Failure rate

The main formula used to compute checkpoint Intervals in HPC systems.

Interval = $\sqrt{2 \times \text{checkpoint-time} \times \text{MTBF}}$

John W. Young, « A first order approximation to the optimum checkpoint interval », Communications of the ACM, Volume 17 Issue 9, Sept. 1974

A more accurate formula by John Daly that integrates restart time.

$$\tau = \sqrt{2\delta(M+R)} - \delta$$

Interval — *MTBF* — *Rst time* — *Ckpt time*