SimHEC: Understanding Application Efficiency at Exascales through Simulations

Da Zhang
Department of Computer Science
Illinois Institute of Technology
dzhang32@hawk.iit.edu

Ioan Raicu
Department of Computer Science, Illinois Institute of Technology
Mathematics and Computer Science Division, Argonne National Laboratory
iraicu@cs.iit.edu

Abstract

It is expected that our HEC system will enter exa-scale era in decade, which is one thousand times of performance as today's system (tera-scale). At the mean time, many challenges also have been noticed and pointed out, as the size of HEC system increased without some dispensable improving on architecture of today's HEC system, the systems could collapse at exascale, due to functionalities would not be able to complete their duties successfully and down the whole system. One potential problem is that MTTF(mean time to failure) of a HEC system will decrease linearly as the system size increase. We will probably have to face the serious situation that our HEC system might be very un-reliable and no works could be done successfully, due to too frequent failures. The situation could be worse for HEC with parallel file system, even armed with checkpointing to guarantee reliability. In this project, we will study and explore application efficiency toward exa-scale, the results shows that DFS offer a quite better performance by taking its advantage in stable storage speed. The work of this project will measure the application efficiency in both parallel and distributed file system, scale from a small system size to exascale with or without checkpointing, observe the differences between three different workloads: uniform workload with only one job running at anytime, uniform workload with ten jobs running concurrently at anytime, intrepid system's 8 month workload from Argonne National Lab. Exa-scale computing is not yet exist, we were using our Java simulator for our research.

Definitions

$$Efficiency = \frac{AppUptime}{AppUptime + AppDowntime}$$

- AppUptime is time an application spending on valuable computing
- AppDowntime is time an application spending on any things else

Checkpointing is art of the state technic for fault tolerance, guarantees our system's reliability, and help to recover from a failure. However, it introduces additional overhead, consumes an application's valuable computing time.

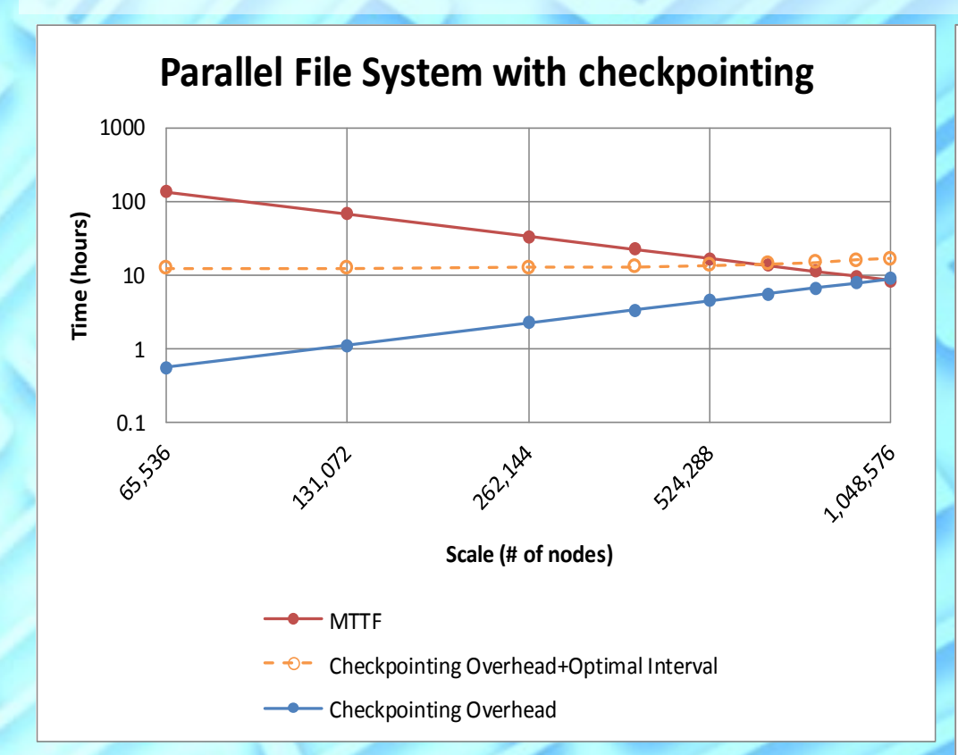
$$MTTF = \frac{MTTF\ per\ node}{num\ of\ nodes}$$

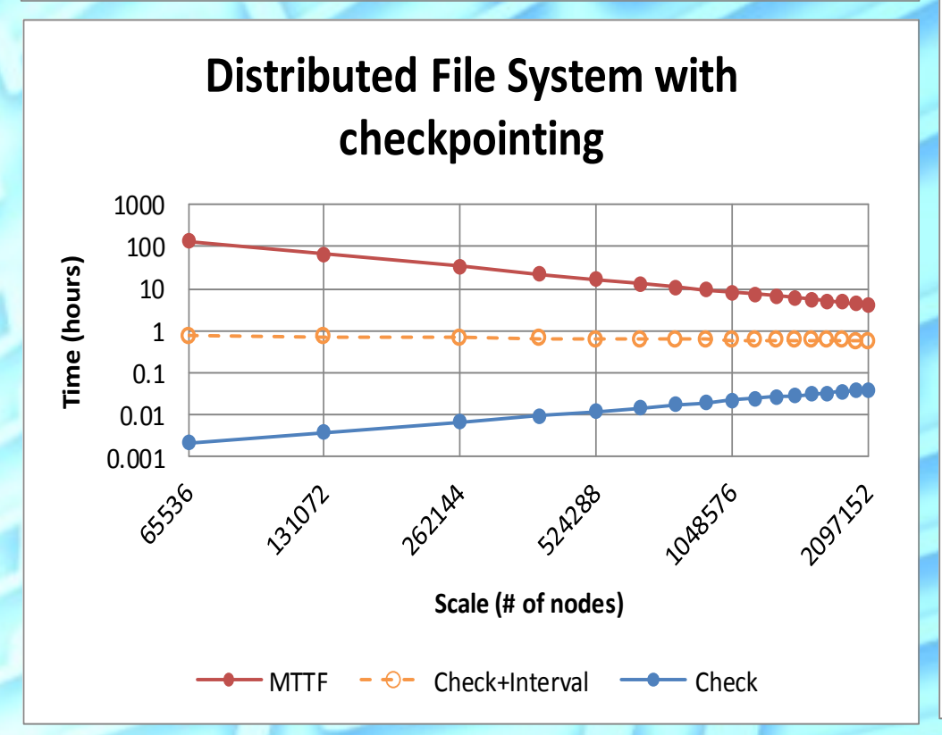
$$Overhead\ of\ checkpointing = \frac{num\ of\ nodes * memory}{speed\ of\ storage}$$

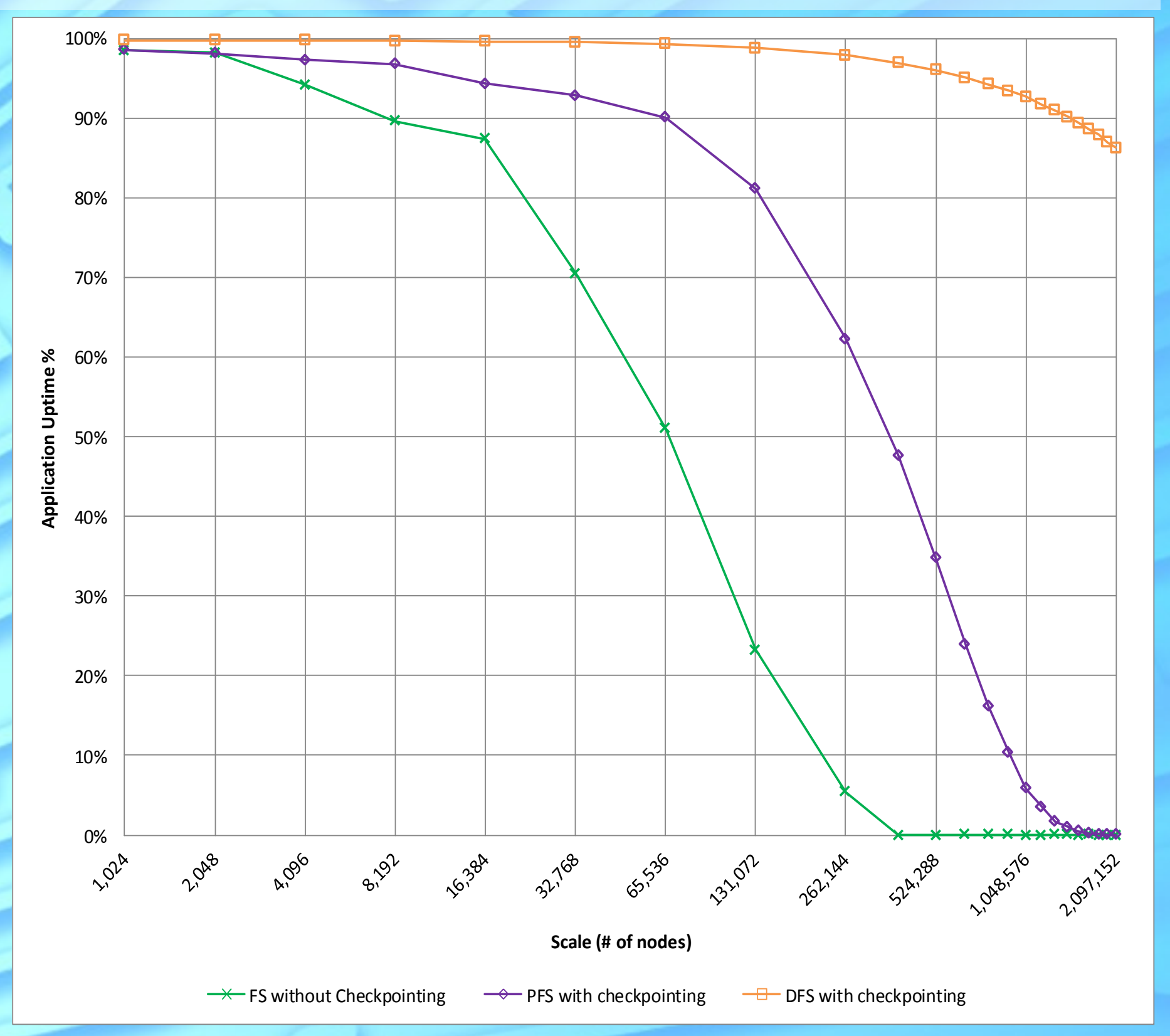
Model for **optimal checkpointing interval**: $t = \sqrt{2\delta(M+R)} - \delta$. t is the optimum checkpointing interval; δ is checkpointing time; M is system MTTF; R is the job's restart time.

Checkpointing Limitations

Checkpointing works greatly now. However, as our system size grow, system MTTF decreases linearly, and checkpointing overhead is also increasing. Once MTTF is becoming so small like hours, even checkpointing might not longer guarantee our system reliability.



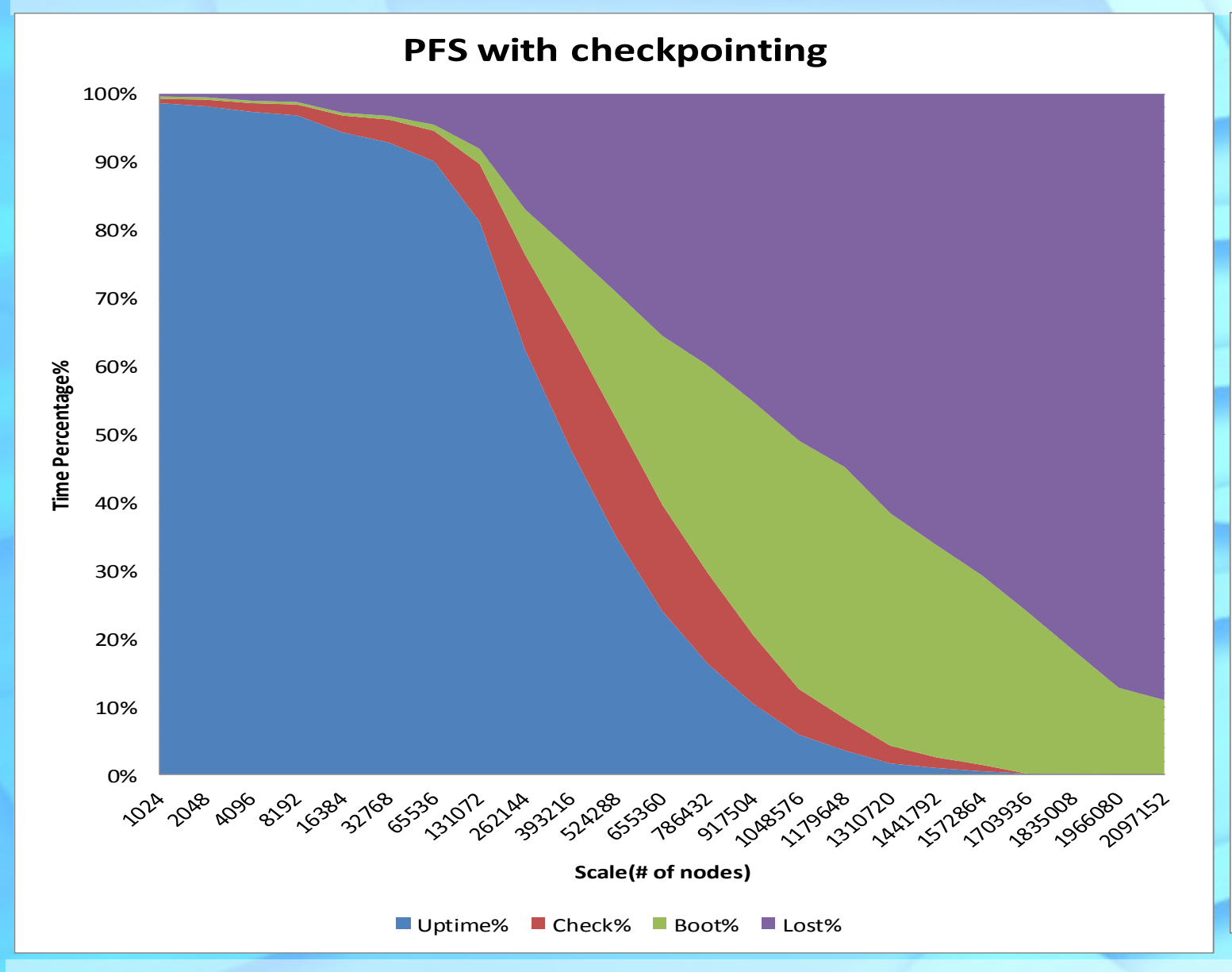


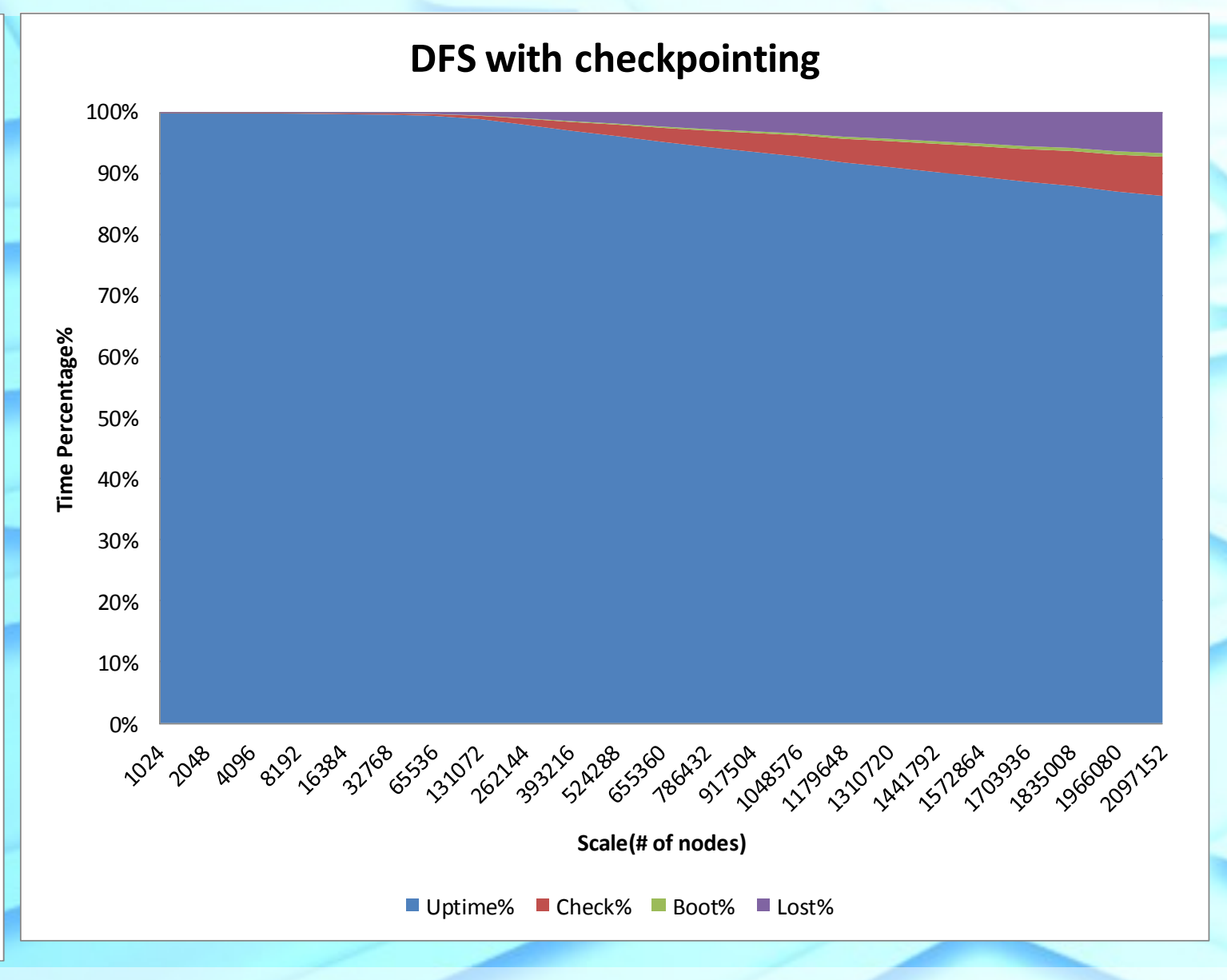


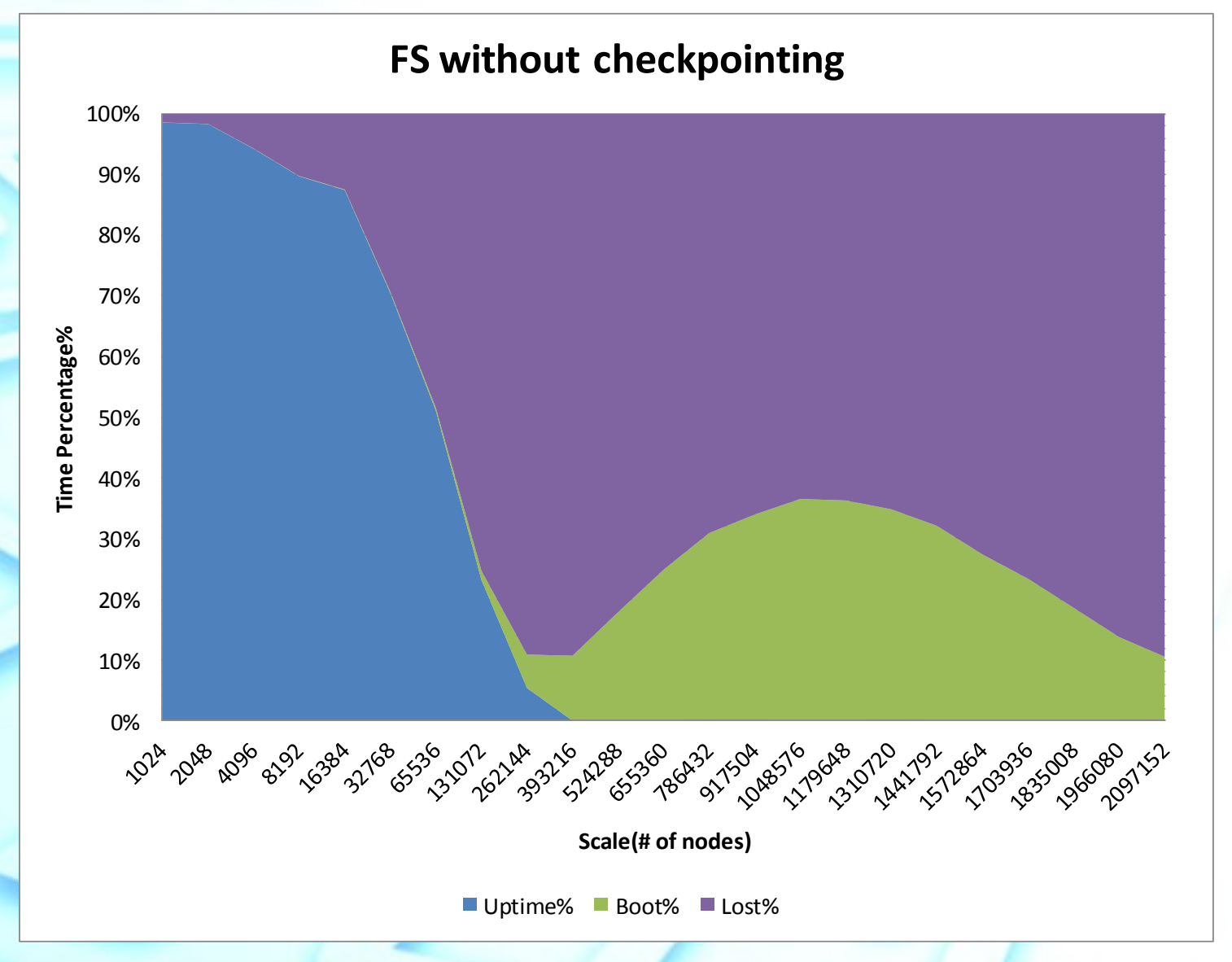
Performance Evaluation

Workload 1:

In the worst case, any time, there was only one big job running, whose size is the same as the system size. In this workload, we keep walltime for each job to be 7 days.

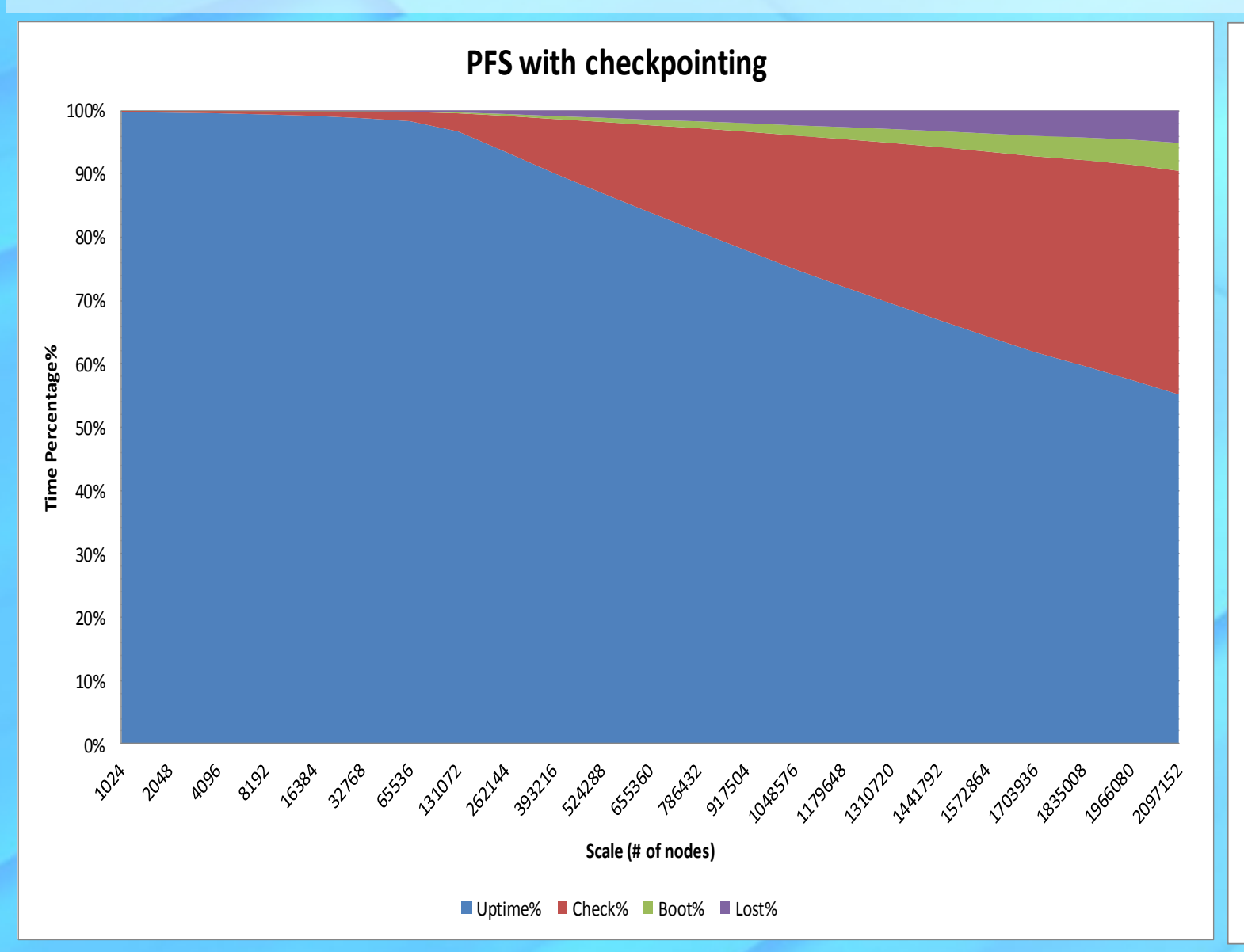


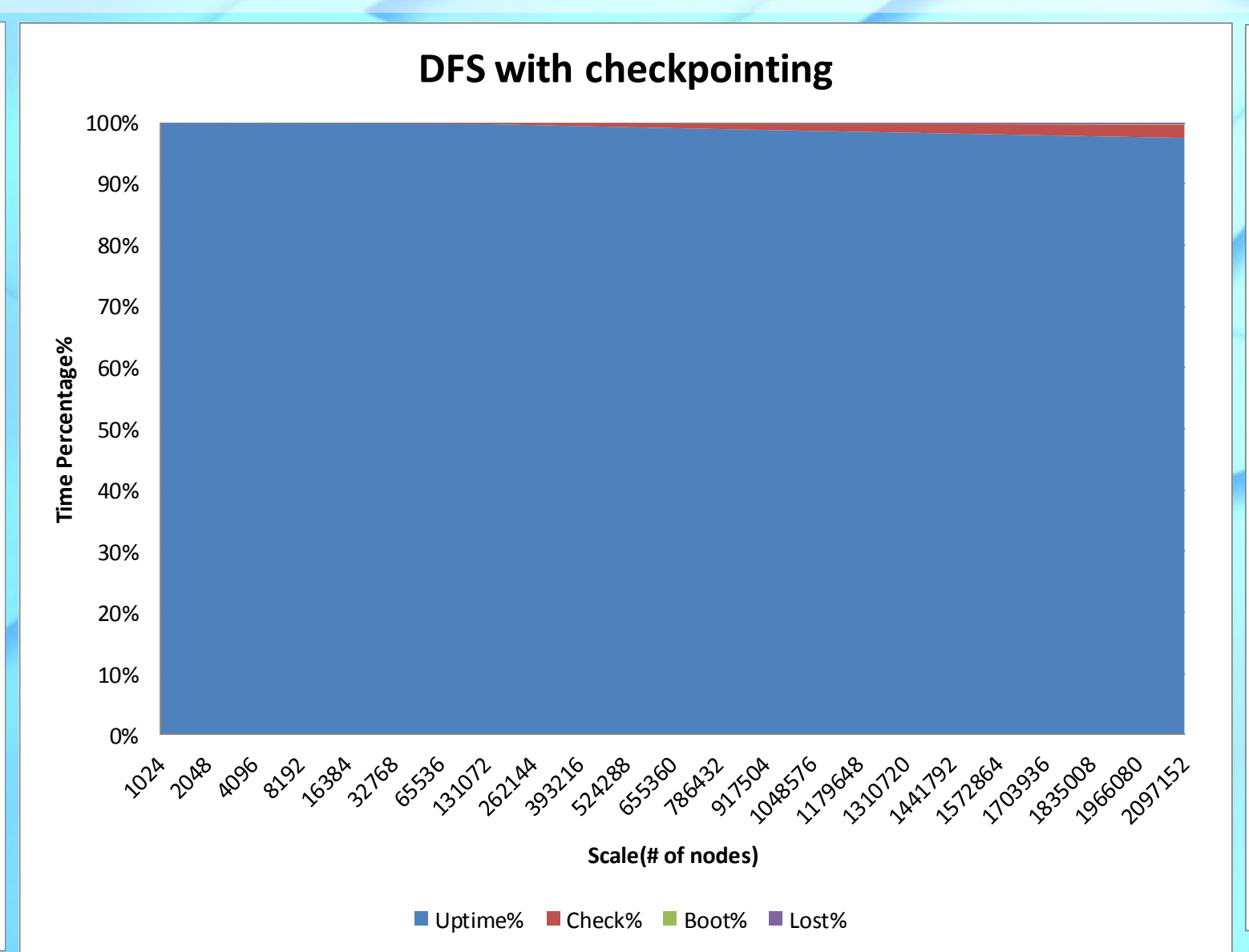


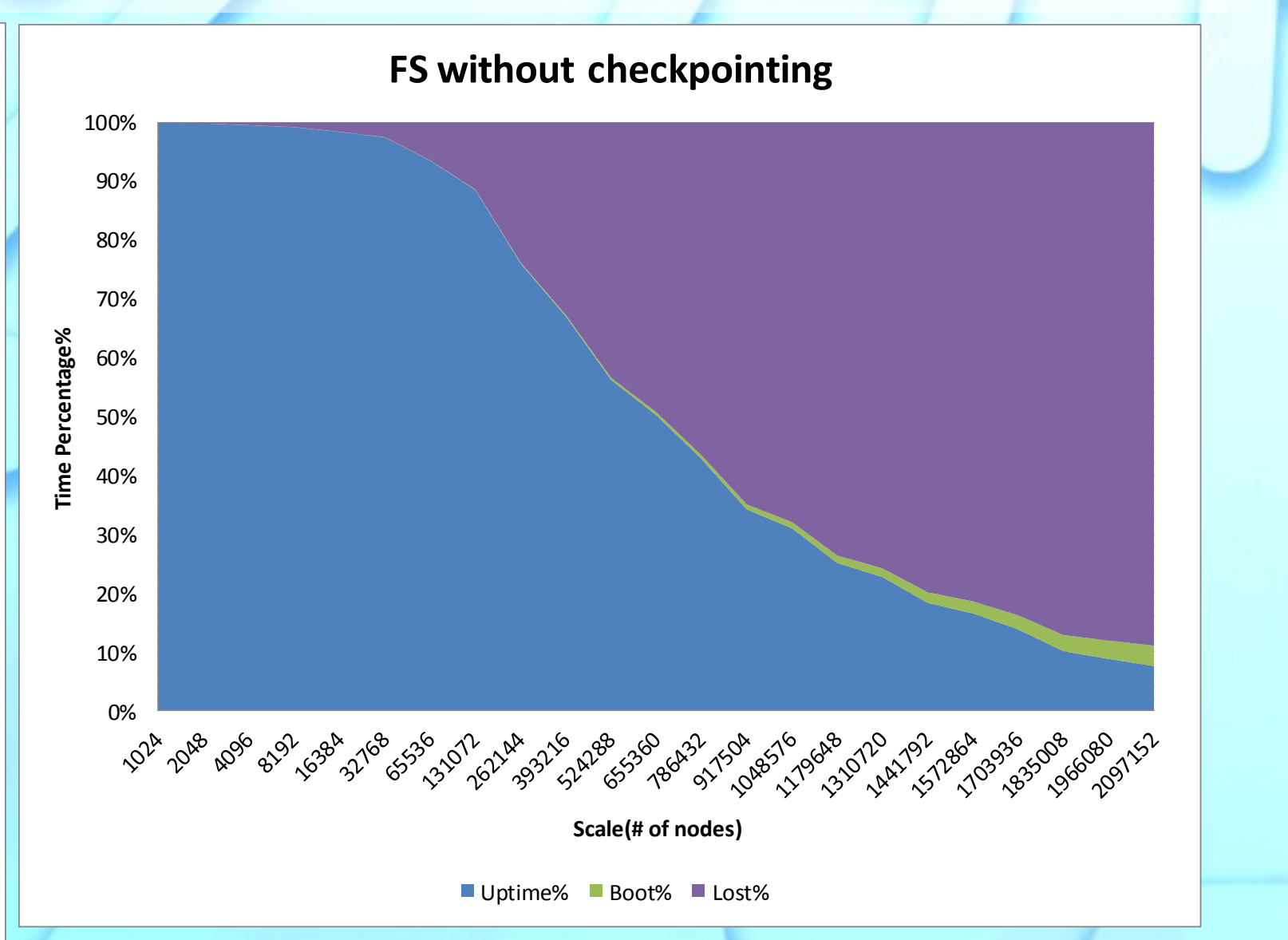


Workload 2:

In multiple jobs run concurrently case, any time, there were ten jobs running, whose size is 1/10 of the system size. In this workload, we keep walltime for each job to be 7 days.

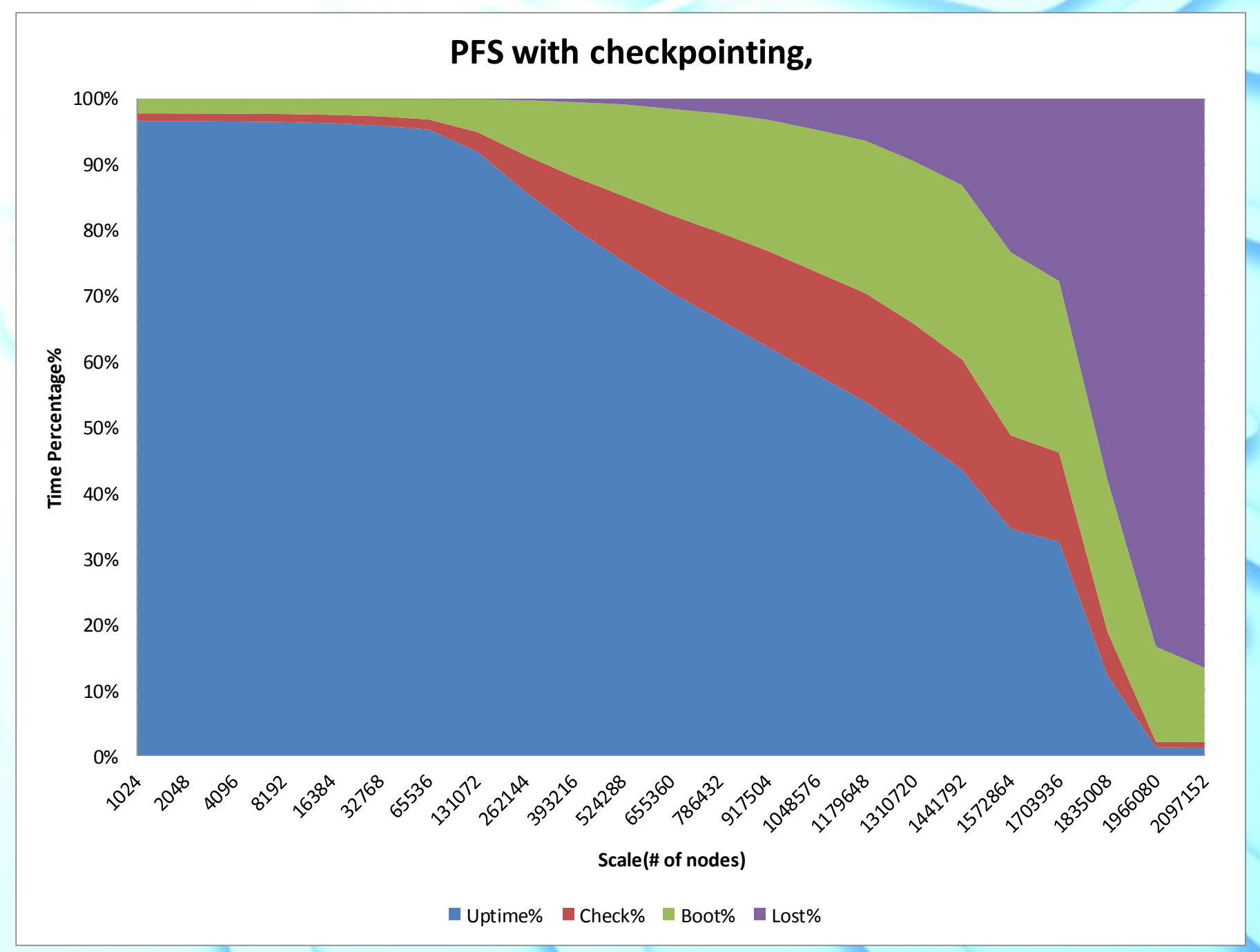


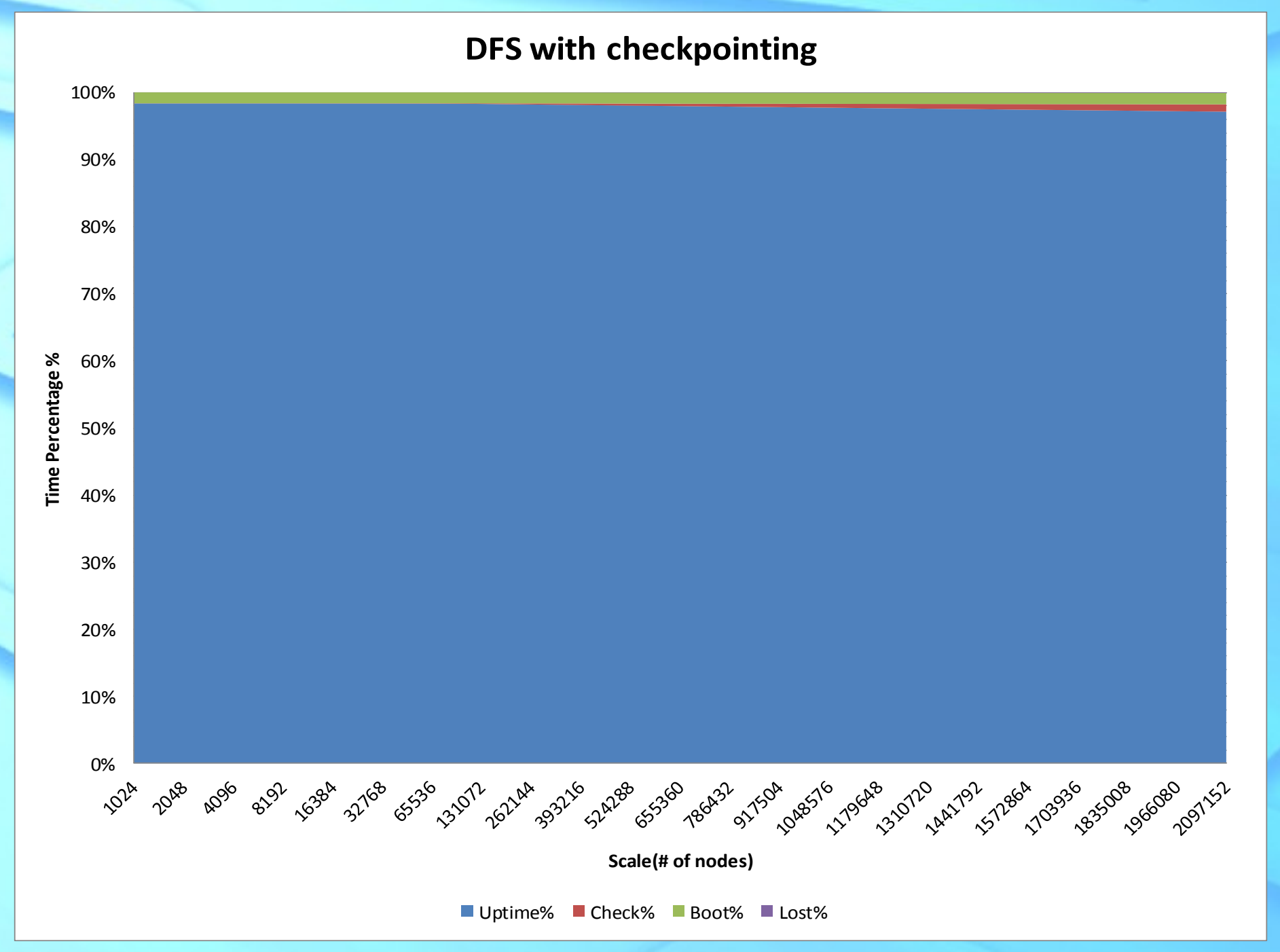




Workload 3:

The workload is Intrepid's 8 month log. Intrepid is the 557TF, 40-rack Blue Gene/P system deployed at Argonne leadership Computing Facility (ALCF) at ANL, comprises 40960 quad-core nodes, with 163840 cores, associated I/O nodes, storage servers, and an I/O network. The log contains Intrepid's 8 month workload, from Jan 2009 to Sept 2009, includes 68936 jobs. In the experiment, we used allocated number of nodes and real running time as request number of nodes and walltime for each job, which may be more than a job's real request. As the system size scale from a small size (1024 nodes) to exa-scale size (2097152 nodes), *job size=(current system size)/(Intrepid system size)×job' s original size*. Walltime of each job isn't changed.





Related material & Acknowledgment

Ioan Raicu,Ian T. Foster, Pete Beckman. "Making a Case for Distributed File Sytems at Exascale", LSAP'11, June 8, 2011.
E,N. (Mootaz) Elnozahy, et al. "System Resilience at Extreme Scale" Defense Advanced Reserch Project Agency(DARPA), 2007.
V. Sarkar, et al."Exascale Software Study: Software Challenges in Extreme Scale System", Exascale Computing Study, DARPA IPTO, 2009.
Workload Log from BG/P at Argonne National Laboratory:
http://www.cs.huji.ac.il/labs/parallel/workload/l_anl_int/index.html
John Daly. "A Model for Predicting the Optimum Checkpoint Interval for Restart Dumps", ICCS 2003, LNCS 2660, pp.3-12, 2003.