Dear reviewers,

Thanks for providing valuable feedbacks. The suggestions, questions and ambiguity raised by the reviewers will be addressed as follows.

**This paper is not considering capital costs for the replication**

When considering future extreme-scale computing, we assume that the hardware resources are fixed (this can be seen in Eq. 4 and Eq. 5), so the CAPEX are also fixed for all fault tolerance alternatives. The difference is how the HW resources are used by each fault tolerance approach. Specifically, Checkpointing/restart uses all CPUs to run main processes, while Process Replication dedicates half resources to replicas. Lazy Shadowing covers the spectrum in between, depending on the nature of failure and energy consumption.

**Can there be a rebalancing stage after a failure so that the shadow sets are reconstituted and all current vulnerabilities are removed?**

Yes, we propose a technique, referred to as shadowed set rejuvenation, to reconfigure the system and eliminate vulnerability. In the case of soft faults, rejuvenation can be accomplished by rebooting the failed cores, and restarting the lost shadows from the state of current mains. For hard faults, it is possible to restart the lost shadows after replacing the failed ones with spares. This will restore a vulnerable shadowed set to its original configuration.

**A core level MTBF is not really appropriate with multi-core processors.**

Similar to [1], we use the term core to represent the computing resource allocation unit (a core, a multi-core processor, or a cluster node), so that our approach is agnostic to the granularity of the hardware platform (Section 3 in the manuscript). We will extend the range of MTBF to 1000 years in the case of core level MTBF for future many-core architecture. Current results show that at MTBF of 1000, Lazy Shadowing outperforms Process Replication by a larger extent, and checkpointing/restart is slightly better than Lazy Shadowing. For example, Process Replication consumers 76.6% more energy than Lazy Shadowing, which consumers 7.0% more energy than checkpointing/restart, when MTBF = 1000 years, W = 10^6 hours, N = 10^6, rho = 0.5. This is consistent with Figure 6. We will update all the figures and analysis in the manuscript after getting all the new results.

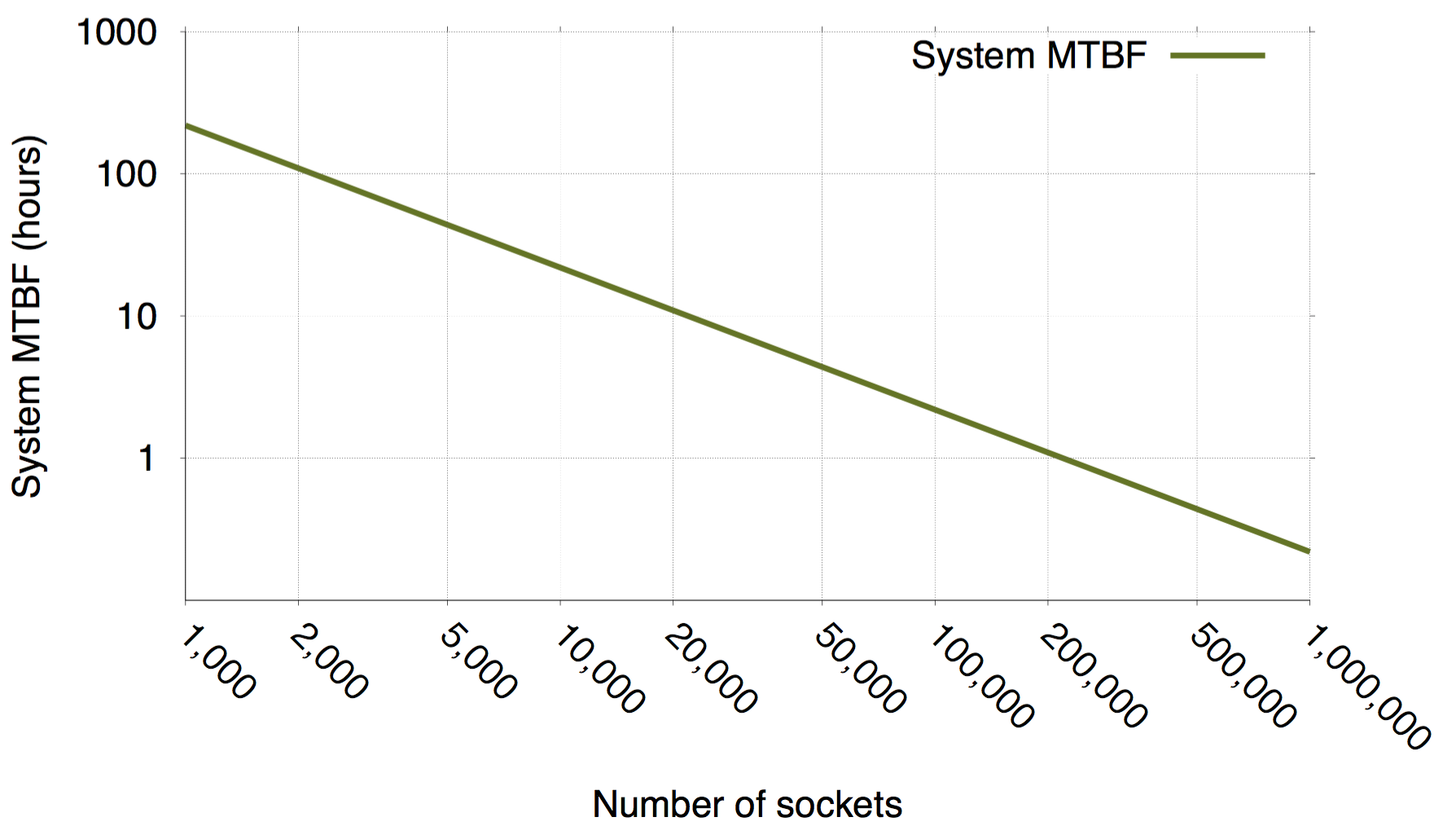
**The claim that traditional CPR times will approach a system's MTBF ignores recent advances in burst buffers. We have nodes that are close to that size (The K computer at 88,000 nodes, BG/Q Sequoia at 120,000 nodes) and all of them currently work with checkpoint/restart.**

Agreed. However, the peak performance of Sequoia is 20,132.7 TFLOPS (K computer is 11,280.4 TFLOPS), which needs about 50x increase to reach extreme-scale performance. Furthermore, as pointed out in [2], the MTBF of these large systems has only been reduced to just a few hours. Lazy Shadowing is an effective alternative to deal with failure at scale, with the unique ability to achieve fine-grained trade-off between completion time and hardware redundancy.

**Figure 1 has problems: too small on high number; lack of time scale; MTBF should be quoted by socket count.**

We acknowledge this limitation, and we will use the following figure () to replace Figure 1 in the manuscript.

Figure 1: Impact of system size on system resilience.

****

**A bit more explanation of the results in 6c and 6d would be helpful. Why is replication better at a low MTBF in 6c?**

Process Replication is better when MTBF is 1 year because the overhead of re-execution for Lazy Shadowing becomes significant when MTBF is extremely low. The likelihood of "application failure" increases as MTBF decreases, and this increases the need for re-execution. Except that, Lazy Shadowing outperforms Process Replication as a result of leaping shadows, which reduce the actual work for the shadow processes. We will add this analysis to Section 6.1.

**Many high end systems today don't at all support swapping or demand paging on the compute nodes and so cannot overcommit their memory.**

The reviewer makes a good and valid point, with respect to overcommitting memory. First, we would like to point out that this is not intrinsic to Shadow Computing, as checkpoint/restart also requires additional memory capacity. Second, we acknowledge the fact that applications and compute kernels in existing HPC environments were simplified significantly by placing a number of restrictions on the application, including eliminating virtual paging and limiting support for OS (Linux) to a handful of system calls. It became clear, however, that strategies designed to work around the capabilities of the hardware cannot scale to exascale computing (See recent DoE reports on exascale computing). Consequently, the research focus has been on new paradigms focused on co-design of hardware with system software to leverage the advantages associated with dynamic, asynchronous mechanisms, such as demand paging and cache tuning, against the design principles and choices of exascale computing. Support of efficient demand paging, through co-design, is particularly critical as it is expected that the data of future exascale applications may not fit entirely in memory [3]. Finally, in comparison to Checkpointing/Restart and pure replication, Lazy Shadow has the capability to control memory usage, based on the nature of failure and existing memory capacity, albeit at a loss of performance.

**What is the impact of replica synchronization, especially with collocation, on failure-free and faulty performance and energy consumption?**

We only perform shadow leaping when a fault occurs and all mains (except the failed ones) are idle during the recovery (Fig. 3). The time for leaping overlaps with failure recovery, and the energy consumption is shown in Eq. 5.

**The work assumes that the main process and the lazy shadow process reside on the same node.**

In Section 3, we state "in order to deal with both permanent and temporary failures, the shadow process starts simultaneously with its associated main process, on a different node." We will further clarify this in the description of Shadow Computing, to remove any potential ambiguity.

**The shadow process would be also subject to failure, and in fact it may fail prior to the main process failing, but this is totally ignored in the paper.**

The shadow process is indeed subject to failure. The underlying assumption of our failure model, however, is that the probability that the main and its associated shadow fail simultaneously is extremely low. Mapping it to the well-known birthday problem, this assumption has been validated by multiple research work, where it was shown that frequency of such an event occurring is much lower than the occurrence of a single node failure [6]. This model was further corroborated by work in [1]. Finally, we would like to point out that the assumption is inherent to Lazy Shadowing, as the model can easily accommodate simultaneous failures of a main and its associated shadow by associating a “suite” of shadows to a given main, where the size of the suite can be determined based on the likelihood of such an event occurring.

**With such message logging, it is entirely possible for a locally checkpointed process to make recovery at full speed using all the resources in the node. So the tossup is between a node that is slower to execute but faster to recover because of the shadow process, or the node that executes at full speed but will have to replay entirely from the checkpoint. But remember the entire story is local.**

The concern raised by the reviewer is precisely why we introduced the concept of "leaping shadows". The proposed approach to deal with this problem is a distributed and coordinated solution. Upon failure of the main process, its associated shadow executes at a higher rate to “catch up” with its main. The remaining main processes, however, become idle as soon as they reach their synchronization barriers. The leaping shadow model takes advantage of this idle period to allow all shadows to leap forward to the same state of their associated main, thereby ensuring forward progress and saving significant computational work (and thus energy).

**It is not clear how M and S are allocated to CPUs/nodes. Can you provide more detail?**

As stated in Section 3, the term core used in the paper is the unit of resource allocation, which may be a CPU core, a processor, or a node, etc. Wherein the case where it means a CPU core, the only restriction is that the two cores for a pair of main and shadow do not reside on the same node, so that a node failure would not affect both of them.

**How are failures detected? What is the cost of failure detection?**

Failure detection, inherent in all fault-tolerant models, is a difficult and challenging problem. In the case of Lazy Shadowing, failures can be detected using a variation of a heartbeat protocol. The cost of failure detection depends on the protocol specifics, such as Heartbeat Interval and Inactivity Count. For simplicity and fairness, since the focus is on the algorithmic behavior of the proposed fault-tolerant computation model, the cost and specific implementation of failure detection was not considered for all fault tolerance approaches covered in this paper.

**What is the empirical support for a MTBF of 25 years?**

Unfortunately, there is no commonly agreed on figure for MTBF. One study even points out that application behavior can affect MTBF [4]. Google assumes 30 years for super reliable machines [5], so we have studied the range of 5 to 25 years.

**The only failure mode is "immediate fail stop".**

Correct. We assume the fail-stop model, which is the same as in Checkpointing/restart and Process Replication (with one replica).

**If a process dies, how does the whole program learn that the shadow process is the new endpoint for all MPI communication?**

Like in MPI, a run-time system is necessary to coordinate all processes for Lazy Shadowing. After detecting the failure of a main process, this system is responsible for replacing the old endpoint with the new endpoint for communication.

**How is I/O handled for the shadow process before the main fails?**

During normal execution, a message-logging protocol is used to ensure consistency between the main and the shadow. Besides, shadow processes remain mute, in the sense that all outgoing messages from shadows are suppressed. More detailed discussion is available in Section 4.3 in the submitted manuscript.

**Lack of implementation**

We implemented an MPI-based prototype, which can execute HPC workloads with lazy replicas [7]. It was demonstrated that lazy shadowing effectively reduces overall energy, and the savings are application dependent. However, since the goal of this paper, which was submitted to the algorithm track, is to introduce algorithmic extensions of the shadow replication paradigm by discussing the novel concepts of shadow collocation and leaping, we did not include a discussion about the implementation. We will include a section in the paper to summarize the basic details and findings of the implementation.

Other suggestions or corrections are all accepted and will be reflected in the revised manuscript.

**References:**

[1] H. Casanova, Y. Robert, F. Vivien, and D. Zaidouni. Combining Process Replication and Checkpointing for Resilience on Exascale Systems. Rapport de recherche RR7951, INRIA, May 2012.

[2] Cappello, Franck, et al. "Toward exascale resilience." International Journal of High Performance Computing Applications (2009).

[3] Cox, Michael, and David Ellsworth. "Application-controlled demand paging for out-of-core visualization." Proceedings of the 8th conference on Visualization'97. IEEE Computer Society Press, 1997.

[4] Sarood, Osman, Esteban Meneses, and Laxmikant V. Kale. "A ‘cool’way of improving the reliability of HPC machines." High Performance Computing, Networking, Storage and Analysis (SC), 2013 International Conference for. IEEE, 2013.

[5] Dean, J. Designs, Lessons and Advice from Building Large Distributed Systems. Available online:  http://www.lamsade.dauphine.fr/litwin/cours98/CoursBD/doc/dean-keynote-ladis2009\_scalable\_  distributed\_google\_system.pdf (accessed on 8 August 2014)

[6] Kurt Ferreira, Jon Stearley, James H. Laros III, Ron Oldfield, “Evaluating the Viability of Process Replication Reliability for Exascale Systems”, SC11, November 12-18, 2011, Seattle, Washington, USA.

[7] Mills, Bryan, Taieb Znati, Rami Melhem, Kurt B. Ferreira, and Ryan E. Grant. "Energy consumption of resilience mechanisms in large scale systems." In Parallel, Distributed and Network-Based Processing (PDP), 2014 22nd Euromicro International Conference on, pp. 528-535. IEEE, 2014.