## EECE 513: Examples of course projects (Term 1, 2011)

The goal of this document is to provide examples of projects that can be done for the course. This does not mean that your project has to be along similar lines – in fact, you are strongly encouraged to choose a project that is in line with your interests.

I have considered three types of projects here. The first type is theoretical or modeling-based, the second type is based on empirical assessment of an existing system and the third type is an implementation of a fault-tolerance technique.

### Example 1: Modeling fault-tolerance techniques for large-scale supercomputers

**Motivation:** Supercomputing systems consisting of hundreds of thousands of nodes are expected to revolutionize scientific computing in the near future (for example, the blue waters supercomputer at UIUC will deliver 1 PetaFlop when it is finished [1]). However, the presence of such large numbers of nodes makes failure a frequent event. Future supercomputing systems must be capable of tolerating frequent failures without disrupting the application's execution. Coordinated checkpointing and recovery has been the de-facto technique for recovering from failures in supercomputing systems. However, studies have shown that these techniques do not scale to large supercomputing systems once the number of nodes crosses a few thousand [2]. Therefore, new techniques are needed to tolerate node failures.

**Goals:** The goal of this project is to evaluate the suitability of fault-tolerance techniques to future supercomputing systems. You will be expected to model a variety of detection, diagnosis and recovery techniques and evaluate their trade-offs under a wide range of fault rates and application characteristics. The main challenge is to come up with a model that can accurately represent the characteristics of the technique, and yet be solved in a reasonable amount of time. You can use modeling tools such as SHARPE or Mobius for this purpose.

**Skills required:** Excellent modeling skills, some experience with parallel systems.

### References:

[1] http://www.ncsa.illinois.edu/BlueWaters/

[2] Wang, L., Pattabiraman, K., Kalbarczyk, Z., Iyer, R.K., Votta, L.G., Vick, C.A., Wood, A.: Modeling Coordinated Checkpointing for Large-Scale Supercomputers. In DSN(2005), pages 812-821.

### Example 2: Empirical assessment of the fault isolation provided by virtual machines

**Motivation:** In the recent past, virtualization has emerged as an attractive technology for workload consolidation and secure computing. Yet, very few studies have rigorously evaluated the fault isolation capabilities of virtual machines (VMs). While virtualization can ensure that failures of one VM do not bring down the whole machine, it is possible for faults in one VM to propagate to another VM due to bugs in the virtual machine monitor's (VMM) implementation. Such bugs can also compromise the reliability and security of the VMM itself, and lead to its failure [1].

**Goals:** The goal of this project is to empirically assess the fault isolation capabilities of a VMM such as Xen [2] by injecting realistic faults in one VM and checking whether the fault propagates to another VM. Because we are injecting faults in the VM, it is possible for the fault to propagate to the Xen hypervisor itself and make it unstable. Therefore, the monitoring must be performed external to the VMM. An important challenge in this project is to determine the kinds of faults that are likely to propagate based on an understanding of the hypervisor's code.

**Skills:** Excellent programming skills, and ability to debug operating systems code. Also, prior experience with Xen or some other hypervisor would be a plus.

### References:

[1] Le, Gallagher and Tamir, Challenges and Opportunities with Fault Injection in Virtualized Systems, First Inter national Wor kshop on Virtualization Performance: Analysis, Characterization, and Tools Austin, Texas, A pril 2008.

[2] Barham et al., Xen and the art of virtualization, Symposium on Operating Systems Principles (SOSP), 2003.

### Example 3: Implementing software rejuvenation of long-running applications

**Motivation:** Software rejuvenation is a pro-active fault-tolerance technique that periodically resets a system to prevent accumulation of corrupted state or leaked resources due to software aging [1]. However, software rejuvenation requires careful engineering of the system to allow such periodic reboots without service disruption. A significant challenge in software rejuvenation approaches is to identify points in the application's execution to perform the rejuvenation. This is especially important in long-running server applications which go through periods of high and low activity. Performing rejuvenation during low activity periods is preferred as the amount of state that needs to be recreated is small [2]. Therefore, one must monitor the application in order to identify the points at which to perform the rejuvenation.

**Goals:** The goal of this project is to implement a software rejuvenation approach for a long running application such as a web server. Because determining the software rejuvenation points is an important part of the project, you must first build a model the web server by observing its activity during a period of time. Based on this model, you need to choose the rejuvenation points to minimize server downtime. Finally, you need to measure the efficacy of your solution by emulating software aging and resource leaks.

**Skills:** Moderate software implementation skills, some performance modeling experience and strong experimental evaluation skills.

### References:

[1] Y. Huang, C. Kintala, N. Kolettis, and N. D. Fulton, "Software Rejuvenation: Analysis, Module and Applications", in Proc. of 25th Symposium on Fault Tolerant Computing, FTCS-25, pages 381–390, Pasadena, California, June 1995.

[2] Y. Bao, X. Sun and K. Trivedi, "Adaptive Software Rejuvenation: Degradation Models and Rejuvenation Schemes," in Proc. of The International Conference on Dependable Systems and Networks, DSN-2003 June 2003.