Chameleon: A Software Infrastructure for Adaptive Fault Tolerance

Center for Reliable and High-Performance Computing
University of Illinois at Urbana-Champaign
1308 W. Main Street, Urbana, IL 61801.
E-mail: {bagchi, kwhisnan, kalbar, iyer} @ crhc.uiuc.edu

1 Introduction
This paper presents Chameleon, an adaptive software infrastructure for concurrently supporting different reliability levels in the same networked environment. Traditionally, fault tolerance has been provided through dedicated hardware, dedicated software, or a combination of both. Hardware solutions from manufacturers like Tandem have provided dedicated fault-tolerant machines with extensive hardware redundancy. Unfortunately, such solutions offer static levels of fault tolerance that remain fixed throughout the lifetime of the system. Software solutions, employed in distributed environments, involve replication of services in software to provide the requisite reliability level. However, to benefit from such solutions, applications need to be written with an intent to run in such an environment. Therefore, the benefits of such middleware go unnoticed to off-the-shelf applications. In contemporary networked computing systems, a broad range of commercial and scientific applications, with potentially varying reliability requirements, need to coexist. It is neither cost effective nor feasible to provide dedicated platforms for hardware-based fault tolerance for each application, or to rewrite each application to leverage off the specialized software middleware. We propose Chameleon as an infrastructure to provide adaptive levels of dependability to off-the-shelf applications with off-the-shelf unreliable hardware.

2 Overview
Chameleon provides the runtime environment for reliability through the use of ARMORs – Adaptive, Reconfigurable, and Mobile Objects for Reliability. ARMORs are components that control all operations in the Chameleon environment and can be classified into three categories: Managers. Managers oversee other ARMORs and recover from failures in their subordinates.
Daemons. Daemons provide error detection for local ARMORs. They also serve as the gateway for communication with any host in the environment. Any participating node requires a locally installed daemon.
Common ARMORs. Common ARMORs implement specific techniques for providing application required dependability, e.g. a Voter ARMOR for voting on replica outputs.
To initialize the Chameleon environment, an entity called the Fault Tolerance Manager (FTM) is installed on an arbitrary node. This is the highest-ranking manager of the manager class. It controls the execution of an application in Chameleon on remote nodes through the daemons that it installs. The user invokes the services of Chameleon through the FTM. He submits the application along with a specification of the application’s reliability requirements. The FTM interprets the specification file and decides on the appropriate fault tolerance execution strategy (FTES) for the application. Then the FTM remotely installs a Surrogate Manager (SM) to handle all activity associated with the application which installs other ARMORs needed to realize the FTES. For example, a SM for Triple Modular Redundant (TMR) FTES would install three instances of a common ARMOR called the Execution ARMOR. The Execution ARMOR oversees the execution of each application replica and forwards the results of the execution for subsequent processing (say, voting). The results are finally collected by the concerned SM, forwarded to the FTM, which forwards it to the user.
Error detection and recovery. Every Chameleon entity as well as user application is overseen by at least one other Chameleon entity and error detection triggers recovery by the entity’s manager. The application is overseen by the Execution ARMOR, the locally installed common ARMORs by the Host Daemon, the SMs by the FTM, and the Daemons by the Heartbeat ARMOR invoked by the FTM. FTM failure promotes itself to be the FTM. ARMOR Reconfigurability and Reusability. A key feature of Chameleon is the ability to create reusable FTESs and reusable ARMORs. For example, the same Execution ARMOR may be used in several different FTESs, and the same FTES may be used with several different user applications. Furthermore, an ARMOR may be reconfigured dynamically to adapt to changing dependability requirements at run-time. This is possible because ARMORs are composed of elementary building blocks. These may be replaced at run-time using a Composition Controller that is a core component of every ARMOR.

3 Conclusion
An early prototype implementation of Chameleon capable of supporting the dual and the TMR modes is available on a
network of Solaris and HP-UX workstations. A sample application we have tested is a distributed matrix multiplication application. Chameleon provides a powerful and flexible framework through which FTES and ARMORs may be constructed and reused to provide varying dependability levels to substantially off-the-shelf applications without the need for specialized hardware.