A Framework for Highly-Available Session-Oriented Internet Services

Bhaskaran Raman and Prof. Randy H. Katz, {bhaskar,randy}@cs.berkeley.edu
CS/EECS Department, U.C.Berkeley, CA 94720, U.S.A.

Robustness and continued service to the end-user in the face of failures are crucial for Internet services. Existing methods of replication and dynamic server selection are targeted for short-lived sessions (most web transactions) – these deal only with the initial choice of a server instance. For services with long-lived client sessions, monitoring the performance and liveness of the initially chosen server instance is important for end-to-end performance. Applications can benefit from dynamic **session-transfer** from one server instance to another when there is poor performance or failure. Examples of such applications include: audio/video transcoding proxies, Internet video-on-demand, and game servers.

Building highly-available **session-oriented** services requires monitoring and fail-over mechanisms – this is the focus of our work. Our goal is to provide a framework for detecting and handling failures at different levels: process/machine failure, site-failure, or network partition. For interactive applications, the sessions have to be re-instantiated *quickly* on failure so that the user sees minimal interruption – this is an important consideration.

In this work, we consider only *stateless* services – those that do not build up state during the session. The examples given earlier fall under this category. Although the monitoring framework will be useful for "stateful" services as well, moving live-sessions has different semantics when there is state involved – session transfer would imply state transfer.

Current cluster-based approaches to high availability do not consider session-transfer for long-lived sessions. There exists no current mechanism for monitoring *distributed* service replicas.

In our framework, service clusters are deployed by independent service providers and are distributed over the wide-area – clusters are the unit of replication. They form peering relationships amongst themselves. *Peer clusters* monitor each others' performance and liveness and serve as backups for one another. Such monitoring is done using periodic heart-beats, and timeouts are used to detect failures at the cluster level. Furthermore, each cluster collects information about the performance of its clients. The session is transferred to a peer cluster on detecting poor performance, or a cluster/network-level failure.

To handle process and machine-level failures, known intra-cluster monitoring mechanisms are used. The advantage of this approach is that the more common failure modes (process/machine failures) are handled within the tightly controlled cluster environment. Another advantage of this cluster-based peering model is that the monitoring overhead (in terms of heart-beats) is amortized over an entire cluster. Furthermore, a cluster environment allows aggregation of client performance information based on parameters like client location – this can then be used to decide the "best" peer-backup-cluster for a given client.

There are several challenging issues in the context of this framework that we plan to explore. There needs to be a way for independent clusters to discover each other, and use policy and network topology information to decide to peer. With respect to the monitoring mechanisms, the challenge is to design one which can detect failures quickly, and do so reliably – without false alarms.

The requirement for robust session-oriented services arose in the context of transcoding services for enabling communication between heterogeneous devices in a hybrid network (the ICEBERG project – http://iceberg.cs.berkeley.edu/). Communication services have especially stringent requirements on robustness and resilience to failures – as exemplified by the telephone system today. We have a prototype transcoding service – we are currently implementing the proposed mechanisms on top of this to study the trade-off between service re-instantiation time, monitoring overhead, and the amount of backup pre-provisioning required.