A. Arpaci-Dusseau **Department of Computer Science**
CS739: Distributed Systems **University of Wisconsin, Madison**

Manageability, availability and performance in Porcupine: a highly scalable, cluster-based mail service – SOSP'99

1 Introduction

- What were the goals of the Porcupine mail server?
- At a high-level, what are four features of the Porcupine system that enable it to meet these three goals?
- Previous related work investigated building scalable web and proxy servers froms clusters. What is more challenging about mail as a service? What two options for placing data (and the corresponding work) were previously investigated for clusters? What are the problems with these approaches?

2 System architecture overview

- One of the keys of Porcupine is that it differentiates hard and soft state. What is the definition of each? What is the benefit of differentiating?
- What are each of the different Porcupine data structures? are they hard or soft state? where is the data stored (is it replicated)?
- Can you walk through Figure 2?
- How does someone *send* mail to a user hosted by Porcupine?
- How does a user *retrieve* messages from Porcupine? What happens if a node holding a mailbox fragment is unavailable? What happens if a user manager is down?

3 Self management

A primary goal of Porcupine is to deal automatically with diverse changes, including node failure, recovery, and addition.

- How does Porcupine determine which nodes are currently part of the service (i.e., how does the Three Round Membership Protocol work)? Why are Lamport clocks used by the membership protocol?

- What different events trigger Porcupine to run the membership protocol? Is it possible for the cluster to be partitioned into multiple groups? How will this look to the user? Can a node believe it is part of group, but it is not? What will happen?
- Do you think the TRM protocol is a good match for Porcupine?
- How is user management assigned to nodes of the system? What is the goal when performing this assignment?
- The user manager node is responsible for two pieces of soft state: message fragment list and the user profile soft state. How is the message fragment list reconstructed? How is the user profile soft state reconstructed?
- How does Porcupine decide which node is responsible for the user profile database itself (hard state)?
- When a new node is added, how does it get used? What data is allocated to it?

4 Replication and availability

- Porcupine replicates the hard state of user database and mailbox fragments to improve availability. In updating the replicas, Porcupine leverages weaker semantics that are specific to mail delivery services. For example, the same message may be received more than once, a message that was deleted may temporarily reappear, and multiple agents acting for the same user may have different views at the same time. Making these assumptions simplifies system design, while improving availability and performance. How can each of these odd cases occur?
- Why are wall clocks, instead of Lamport clocks, used to synchronize updates to the replicated user database?
- Assuming no failures, what is the protocol for updating a replicated object? What is the purpose of the log? Why would a peer need to keep a log too? What is the performance problem with keeping a log? (At what point may the coordinator respond to the initiating agent??)
- If a node with a replicated mailbox fragment disappears, how is another replica made???

5 Dynamic load balancing

- Load balancing is performed at the level of individual message sends. How does a node determine the load on another node? Is this a good approach for Porcupine?
- What two tensions must be resolved when deciding where to place a new message? How does Porcupine decide on which node to store a new message?

6 System evaluation

- How well does the performance of Porcupine scale up through 30 nodes? How much worse does replication perform? Why so much worse?
- Why does dynamic load balancing perform significantly better than static load balancing given skew in the workload? Do you think the authors' concern with high spread factors is warranted?
- Given a heterogeneous configuration (some machines have faster disks), why does dynamic load balancing help more?
- Failure recovery seems to work, as shown in Figures 10 and 11.
- Do you think they demonstrated availability, performance, and manageability? Are there other experiments you would have liked to have seen?

7 Conclusions

- Great example of a system service tuned to its particular workload and assumptions. Took advantage of soft state and email semantics to simplify design, improve availability, performance, and manageability.