Distributed Object Capabilities

Simon Gerber[*], Timothy Roscoe; ETH Zurich
{simon.gerber,troscoe}@inf.ethz.ch

September 29, 2016

Abstract

Modern computers are rapidly increasing in size. This leads to various problems, one of which is that it is getting increasingly harder to keep track of who owns what resources in the system. Object capabilities [3, 4, 6] are a well-established method for tracking resource ownership. Most of the previous work on object capabilities has been done on small systems and the operation of such a capability system generally relies on a centralized database which keeps track of all the existing capabilities. That centralized database does not scale to a modern, rack-scale computer system and needs to be distributed across the nodes of such a system.

The operations that can be performed on capabilities are: *invoke*, which allows an application to perform an operation on the object referenced by the capability, *copy*, which creates a new capability that references the same object as an existing source capability, *delete*, which deletes the capability, *retype*, which creates a new capability, which we call a descendant, with a type that is derived from the type of an existing source capability, and *revoke*, which finds and deletes all copies and direct descendants of a capability. The capability operations that need synchronization, which we also call *expensive* operations, are *retype*, *delete*, and *revoke*. The other operations, *invoke*, and *copy*, we call *cheap* operations.

Our main contributions are: (1) a fast per-node index for local capability lookups based on a number of query parameters, and (2) a set of distributed algorithms for the operations mentioned above. This work is based on work done in the scope of a master's thesis in our group [5].

The per-node capability index, which is needed to make local capability operations fast, is an index over all the capabilities in that node's capability database. The index is implemented as an AA tree [1], which is an isomorphism of a 2–3 tree. The AA tree is a variation of the red-black tree where red nodes can only be right subchildren. This preserves the red-black tree property that the deepest leaf is at no more than twice the depth of the shallowest leaf, and further guarantees that the deepest leaf is the rightmost element in the tree. We make heavy use of the index when processing expensive operations, as all of the expensive operations involve capability lookups on all nodes. Those lookups are used to look for capabilities that refer to the object on which the operation is executed or objects that were derived from that object.

Expensive operations are synchronized by electing one node in the distributed system as the leader and serializing all the operations through that leader. The leader also coordinates operations that involve multiple nodes in the system. Notably, the leader node does not need to be the same for all capabilities, and in order to load-balance the system it is preferable to make all nodes leaders for a subset of the total set of capabilities. If a non-leader node N wants to perform an expensive operation on a capability it will contact the leader node L which will perform the operation on behalf of N.

Additionally, leadership can be transferred between nodes, either voluntarily, or when the last capability to an object that exists on the current leader node is deleted. To elect a new leader in the latter case, we arbitrarily pick another node that holds a capability to the object to become the new leader.

We have a working prototype implementation of this distributed capability design at a relatively small scale at the operating system level in Barrelfish [2]. Barrelfish uses segregated capabilities where the capability metadata is stored in special capability regions in memory. We use a custom implementation of the AA tree whose nodes are stored along-side capability metadata. Each distributed algorithm is implemented partly in the privileged-mode kernel of Barrelfish, the so-called *cpu driver*, and partly in the user-mode kernel of Barrelfish, the *monitor*, due to Barrelfish's multikernel design where cpu drivers cannot directly communicate with each other.

[*] student

References

[1] A. Andersson. Balanced search trees made simple. In *Proceedings of the Third Workshop on Algorithms and Data Structures*, WADS '93, pages 60–71, London, UK, UK, 1993. Springer-Verlag.

[2] A. Baumann, P. Barham, P.-E. Dagand, T. Harris, R. Isaacs, S. Peter, T. Roscoe, A. Schüpbach, and A. Singhania. The multikernel: a new OS architecture for scalable multicore systems. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, SOSP '09, pages 29–44, New York, NY, USA, 2009. ACM.

[3] N. Hardy. KeyKOS architecture. *SIGOPS Oper. Syst. Rev.*, 19(4):8–25, Oct. 1985.

[4] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, and S. Winwood. seL4: formal verification of an OS kernel. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, SOSP '09, pages 207–220, New York, NY, USA, 2009. ACM.

[5] M. Nevill. An evaluation of capabilities for a multikernel. Master's thesis, ETH Zürich, May 2012.

[6] M. V. Wilkes. *The Cambridge CAP Computer and Its Operating System (Operating and Programming Systems Series)*. North-Holland Publishing Co., Amsterdam, The Netherlands, The Netherlands, 1979.

Distributed Object Capabilities

Simon Gerber, Timothy Roscoe
{simon.gerber,troscoe}@inf.ethz.ch, Systems Group, ETH Zurich

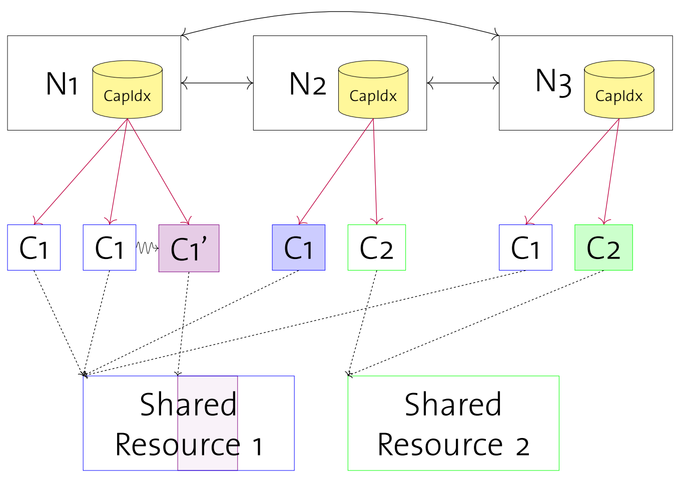
Motivation

- Decentralized resource management at rack scale is hard
- Object capabilities work at small scale
- Centralized design does not scale well

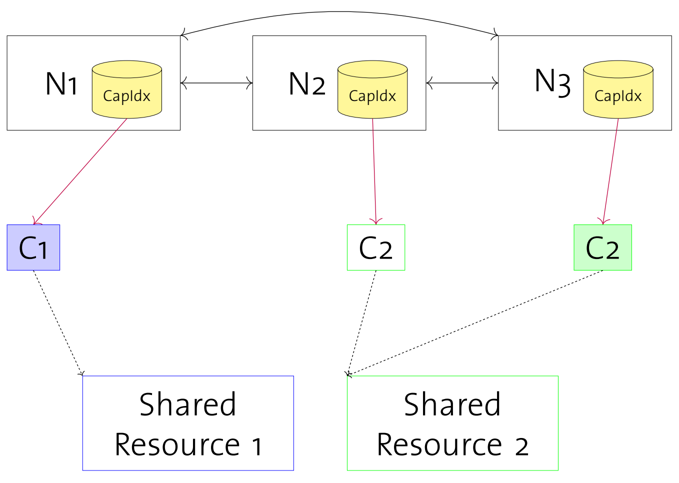
Contributions

- Fast index for capability lookups on single node
- Distributed capability algorithms and database
- Proof of concept implementation in Barrelfish

Example: Revoke



Initial system state



System state after revoking *C1* from *N1*

⟷ message channel	→ capability index entries	🟦 owned capability
⇝ retype relationship	⇢ capability-object relationship	☐ foreign capability

Algorithm Design & Implementation

- Agreement protocol for each object
- One leader node for each object
- Operations are serialized at leader node
- Each node has tree-based index for finding descendants and ancestors of object

Algorithm Invariants

- Every capability that is not Null has a *leader* node
- Any two capabilities that are copies must have the same *leader*
- The leader node for a capability must always have a local copy

Future Work

- Correctness proofs for algorithms
- Performance Evaluation

Conclusions

- Good fit in modern OS
- Useful for decentralized access control

ETH Eidgenössische Technische Hochschule Zürich Swiss Federal Institute of Technology Zurich

inf Informatik Computer Science

Systems@ETH Zürich

www.barrelfish.org