Proposal: Inserting Caches using Compiler Passes to Optimize Passes to Optimize RPCs between Microservices

Adarsh Sreedhar (adarshsr), Bhakti Chaudhari (bchaudha)

https://adarshsreedhar96.github.io

1 Project Description

Acme Air is an in-house project in CMU that employs a Microservice Architecture (MSA) to simulate an airline travel reservation website. Since there are multiple microservices present, such as handling reservations, airline miles, cancellations/rescheduling etc. there will be frequent communication between them, even for a single customer ticket booked, and so network latency is high. To reduce the traffic by the extent possible without providing any stale or incorrect data to the customer, we aim to add few compiler optimizations replacing those expensive RPC calls by a cache.

100% goal
Write a compiler pass in Soot for a Java based Microservice Application (Acme Air) to insert caches for selected RPC calls to achieve an increase in performance while maintaining 100% accuracy by implementing some cache coherency protocols.

75% goal
Write a compiler pass in Soot which is able to correctly generate IR for the application, identify locations in code of RPC calls, and insert a cache to improve performance while ensuring eventual consistency/correctness.

125% goal
On top of the 100% goal, identify the most ideal locations to insert this cache by using dynamic analysis techniques to monitor access patterns in the application and optimize based on those. This will improve overall performance of the application instead of optimizing for specific calls. Possibly develop a generic way to choose the best location for inserting this compiler optimizations.

2 Plan of Attack and Schedule

	Bhakti	Adarsh	Target
Week 0 (14th - 20th March)	Literature Survey	Literature Survey	Complete Literature Survey to understand work that's already been done in the area and
Week 1	Get familiarized with Soot and the use case application Acme Air	Get familiarized with Soot and the use case application Acme Air	Get up to speed with work already done on this project and develop a better understanding of this specific use case. Become familiar with the tools needed to work on this, eg - Soot.
Week 2	Generate IR for Acme Air using Soot	Given a set of IR in Soot, explore ways to identify specific parts of code (like RPC calls) and ignore others (like external library code). Possibly using annotations in code.	Generate IR of Acme Air and develop the ability to identify which parts of the IR we are targeting our optimizations for.
Week 3	Develop a pass to insert instructions to replace chosen RPC calls with instructions to go to cache instead.	Develop a way to generate IR of caching logic which is generic enough to be inserted in place of different RPC calls without much changes.	Initial compiler pass implementation which inserts caching logic in place of chosen RPC calls.
Week 4 (11th - 17th April)	Develop a cache coherency protocol suitable to the application and integrate with the compiler pass.	Combine the work of previous weeks to get a working compiler pass that inserts caches in place of selected RPC calls.	Develop the caching logic for better performance and coherency.
Week 5	Apply this pass on the Acme Air application and run a series of tests to measure performance	Analyze results of tests and apply further improvements and fine tuning for improved performance and correctness.	Final compiler pass optimization which provides 100% correctness and measurable performance improvement.

	and correctness metrics.		
Week 6 (Project Due)	Perform dynamic analysis on the application to collect information on application flow and request patterns.	Figure out how to use the information from the dynamic analysis to identify avenues for optimization.	Identify suitable places to apply the pass for improved overall performance of the application.

3 Milestone

Our milestone goal is to have a compiler pass in Soot which can be applied on a selected microservice in Acme Air to optimize its RPC calls by replacing them with cache accesses. For the milestone, we will implement an expiry-based protocol on the cache to ensure eventual consistency with a fixed time delay.

4 Literature Search

Ziv Scully et al. [1]'s work on adding compiler optimzations to cache SQL query results automatically to web applications without modifying the source code can give us sound ideas on how to create caches with concurrent access. Their work shown on many microbenchmarks promised a double or even higher increase in throughput, all with an extra argument passed to the compiler.

Amit et al [2]'s work on Database Scalability Services, which caches application's query results and answers queries on their behalf, also includes several aspects to consider when thinking about the possible scenarios where cache invalidation or staleness of data is a concern, while also encrypting the data for privacy reasons.

Charlie et al [3]'s work on building a distributed cache can also give us insights on how to maintain coherency across the local caches of different microservices. The publish-subscribe system used in the Ferdinand Architecture is a viable design to borrow ideas from.

5 Resources Needed

Soot (https://github.com/soot-oss/soot) - Open Source

6 Getting Started

We have started with the literature review. We are also getting ourselves acquainted with the Soot framework. As we have only written passes for C/C++ in llvm for this class, we are trying to get familiarized with writing compiler passes for Java using Soot.
We do not have any blockers to get started at the moment.

References

[1] Ziv Scully, Adam Chlipala. A Program Optimization for Automatic Database Result Caching. Proceedings of the 44th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'17). January 2017.

[2] A. Manjhi et al., "Invalidation Clues for Database Scalability Services," 2007 IEEE 23rd International Conference on Data Engineering, 2007, pp. 316-325, doi: 10.1109/ICDE.2007.367877.

[3] Charles Garrod, Amit Manjhi, Anastasia Ailamaki, Bruce Maggs, Todd Mowry, Christopher Olston, and Anthony Tomasic. 2008. Scalable query result caching for web applications. Proc. VLDB Endow. 1, 1 (August 2008), 550–561. DOI:https://doi.org/10.14778/1453856.1453917