Instantaneous Archives

Walter Landry[1] and Serge Monkewitz[1]

[1]*Infrared Processing and Analysis Center, Caltech, Pasadena, CA 91125, USA;* wlandry@caltech.edu

Abstract. IRSA, the NASA/IPAC Infrared Science Archive, is one of the largest and busiest astronomy archives in the world. In the past, our main emphasis was on making new data and new capabilities available. With the widespread implementation of Virtual Observatory protocols, there are a number of useful tools that can quickly and easily perform insightful, sophisticated queries from archives around the world (Taylor 2017; Astropy Collaboration et al. 2013). The queries, if not handled quickly, can easily overwhelm the site and interfere with other users. In addition, reducing latency below the point of human perception enables more interactive and exploratory science.

This paper discusses how we are re-architecting our query pipeline to eliminate process, file system, and database connection overheads. Taken together, these improvements delivered radical, order of magnitude improvements in latency and throughput.

1. Introduction

IRSA, the NASA/IPAC Infrared Science Archive[1], is the official archive for NASA's infrared and submillimeter missions. Archives serve a vital role in NASA missions, with half of all publications from a mission coming from archive users (Rebull 2017). IRSA was started in 1993, and the underlying mechanisms for serving data to users has changed dramatically over that time. The current iteration of the software and hardware architecture was mostly engineered to satisfy two use cases. One is that of users visiting the website and manually entering search parameters. The other is that of users writing simple scripts to perform a modest number of queries. In both cases, returning a result in a few seconds is completely adequate.

That is no longer adequate. Scientists want, and increasingly demand, to be able to perform complex queries over all of the billions of records at IRSA and receive the results immediately. They use programmatic APIs to perform enormous numbers of concurrent queries. It is not uncommon for us to get more than 100,000 queries in a single day. The rapidly increasing size of the datasets from new missions leads to a desire to do more analysis through the web site. That drives a requirement for more sophisticated web interfaces which can generate many queries as the user moves their mouse (Wu 2017). Delays of a few seconds then become an exercise in frustration.

[1]http://irsa.ipac.caltech.edu

Finally, as a part of IPAC, we share our data center with other NASA missions. They would like to use IRSA to support their pipelines and archives, and ideally the time to fetch results would be dictated by data center network latency (~0.3 ms) rather than software.

In this paper, we will discuss IRSA's strategy for dealing with this new reality. To constrain our efforts, we will focus on reducing the time to get a complete response from our most popular service, the IVOA Simple Cone Search (SCS)[2]. When a user queries the IRSA SCS service, the result is a list of objects within a specified radius of the search position. Since releasing the service in the beginning of 2015, it has come to dominate our query volume. To be even more specific, we will look at small cone searches (0.6-60 arcseconds) of random points in the sky of the 2MASS All-Sky Point Source Catalog (Skrutskie et al. 2006). This 2MASS catalog is both large enough, with 471 million objects, to be a realistic test, and small enough to copy around and test without undue difficulty (about 500 GB). For the 2MASS catalog, these queries generally return a small number of results (0-10).

2. Existing Architecture: Apache + CGI

These types of cone search queries have always been very common. So we have optimized our database to handle it, using the Hierarchical Triangular Mesh (HTM) to tile the sky (Groom et al. 2014, Sec. 6.1). With the 2MASS catalog, when executing a query directly against our Oracle 11g database, the median time to get the result is about 30 ms.

However, when fetching the result through the SCS service, even within the data center, we measure much larger median times of 700 ms. The root cause of this disparity lies in our use of the CGI mechanism bundled with the Apache web server. This mechanism starts a new program for each query, which in turn runs a series of other programs. For every query, we fetch metadata about the table as well as the result itself. So for each query, we must make two completely new database connections and several new processes. About 1/3 of the 700 ms query result time is just starting new processes, while the other 2/3 is initiating new database connections. The actual database query time is in the noise.

Additionally, the programs were written when memory was much less plentiful, so they make extensive use of a networked file system to store intermediate results. This does not currently show up as a bottleneck, probably because we have decent hardware driving our storage system. But it is something to keep in mind when optimizing the system.

3. New Architecture: Embedded Web server

Given the description of the problem, the solution is fairly obvious. We can eliminate the cost of new processes by running an embedded web server that spawns new threads to handle requests. This is similar to Java Servlets[3], a technology already in wide use

[2] `http://www.ivoa.net/documents/REC/DAL/ConeSearch-20080222.html`

[3] `https://en.wikipedia.org/wiki/Java_servlet`

among astronomy data centers. This also allows us to keep a pool of connections that the threads can use as needed, instead of constantly creating and destroying them.

Our stack is mostly C++, so our preference is for an embedded web server that nicely integrates with C++. Searches on the web turn up more than 20 projects which might be suitable. Some of them we set aside because of their relative instability, lack of recent activity, significant dependencies, or overly complicated API. For the remaining few candidates, we wrote a simple application which waits for 10 seconds and returns "Hello World". Then we used httperf to make 128 connections per second for a total of 1024 connections[4]. Most of the candidates just broke, while others did not perform well. Given these results, we decided to use libhttpserver[5], a C++ wrapper around libmicrohttpd[6]. Even then, we still had to make a few improvements to libhttpserver to accommodate our needs. Coincidentally, the ALMAWebQL site also decided on libmicrohttpd for their embedded web server (Zapart et al. 2017).

For connecting to the database, there are, again, many choices. We decided to use the official Oracle OCCI library because of its easy, built-in support for connection pools. It is also, presumably, no slower than other alternatives.

The last optimization is to do as much as possible in memory. When a query radius is small enough (less than 1 arcmin), then nothing is ever written to a file system. We load the query result straight into memory, do any necessary processing or file type conversion, and then write directly to the outgoing http connection.

With all of these improvements, median response times drop from 700 ms to 80 ms. This is a dramatic improvement, so we will be deploying this new backend soon.

4. Future Directions

80 ms is still a far cry from instantaneous inside the data center. There are still a few small things we could do at the cost of increased complexity, such as caching the table metadata. But for that effort to be worthwhile, we will have to also radically improve the response times for the raw database query. We have experimented with Postgres using the Q3C (Koposov & Bartunov 2006) and H3C[7] tiling, but the timings were neither significantly better nor worse than the Oracle results.

The results presented by (Korotkov & Bartunov 2017) demonstrated single digit millisecond query times for Postgres+Q3C if the entire table fits in memory. We do not have access to a single machine with enough memory. Instead, we loaded the 2MASS catalog into a MemSQL[8] instance which distributed the table over 10 machines. The results were OK, with raw SQL queries returning in 8 ms. It probably means that, even in optimal conditions, we will not be able to return results in less that 10 ms. This would, at least, be faster than screen refresh rates (~16 ms).

---

[4]Specifically, the command: httperf –hog –num-conns=1024 –num-calls=1 –rate=128 –server localhost –port 8080

[5]`https://github.com/etr/libhttpserver`

[6]`http://www.gnu.org/software/libmicrohttpd/`

[7]`http://cds.u-strasbg.fr/resources/doku.php?id=h3c`

[8]`http://www.memsql.com/`

There are also practical problems with MemSQL. It only solves the problem for that one table, and the expense to replicate this setup for all of our tables would be prohibitive. Also, there are some architectural decisions in MemSQL which result in cross matches against small user uploaded tables taking much longer than our current setup. However, it does suggest that we might be able to get significant improvements if we moved our busiest tables from HDDs to SSDs.

As a final note, we are also looking into clustered databases, such as LSST's QServ[9] or the Citus extension to Postgres[10]. They will help more with handling many concurrent requests rather than reducing the latency of a single request. They also have the added benefits of shorter times to load large tables and faster execution of more complex queries which require full table scans.

Acknowledgments. We would like to thank Mark O'Dell, Angela Zhang, Scott Terek, and Gilles Landais for their assistance.

References

Astropy Collaboration, Robitaille, T. P., Tollerud, E. J., Greenfield, P., Droettboom, M., Bray, E., Aldcroft, T., Davis, M., Ginsburg, A., Price-Whelan, A. M., Kerzendorf, W. E., Conley, A., Crighton, N., Barbary, K., Muna, D., Ferguson, H., Grollier, F., Parikh, M. M., Nair, P. H., Unther, H. M., Deil, C., Woillez, J., Conseil, S., Kramer, R., Turner, J. E. H., Singer, L., Fox, R., Weaver, B. A., Zabalza, V., Edwards, Z. I., Azalee Bostroem, K., Burke, D. J., Casey, A. R., Crawford, S. M., Dencheva, N., Ely, J., Jenness, T., Labrie, K., Lim, P. L., Pierfederici, F., Pontzen, A., Ptak, A., Refsdal, B., Servillat, M., & Streicher, O. 2013, A&A, 558, A33. 1307.6212

Groom, S., Ly, L., Lynn, M., Mi, W., Monkewitz, S., O'Dell, M., Rey, R., Roby, T., Teplitz, H., Terek, S., et al. 2014, in ADASS XXIII, vol. 485, 185

Koposov, S., & Bartunov, O. 2006, in Astronomical Data Analysis Software and Systems XV, edited by C. Gabriel, C. Arviset, D. Ponz, & S. Enrique, vol. 351 of Astronomical Society of the Pacific Conference Series, 735

Korotkov, A., & Bartunov, O. 2017, in ADASS XXVI, edited by TBD (San Francisco: ASP), vol. TBD of ASP Conf. Ser., TBD

Rebull, L. 2017, in ADASS XXVI, edited by TBD (San Francisco: ASP), vol. TBD of ASP Conf. Ser., TBD

Skrutskie, M., Cutri, R., Stiening, R., Weinberg, M., Schneider, S., Carpenter, J., Beichman, C., Capps, R., Chester, T., Elias, J., et al. 2006, The Astronomical Journal, 131, 1163

Taylor, M. 2017, in ADASS XXVI, edited by TBD (San Francisco: ASP), vol. TBD of ASP Conf. Ser., TBD

Wu, X. 2017, in ADASS XXVI, edited by TBD (San Francisco: ASP), vol. TBD of ASP Conf. Ser., TBD

Zapart, C., Shirasaki, Y., Ohishi, M., Mizumoto, Y., Kawasaki, W., Kobayashi, T., George, K., & Eguchi, S. 2017, in ADASS XXVI, edited by TBD (San Francisco: ASP), vol. TBD of ASP Conf. Ser., TBD

[9] https://github.com/lsst/qserv

[10] https://www.citusdata.com/