Final Project: Analyze an actual system

Updates

Fri Apr 5: Added grading rubric and Frequently-Made Suggestions.

Overview

In this final project, you will apply the principles described in this course to analyze an actual system in production. You will practice skills that you would use in a work context:

- Learning the architecture of an existing system you were hired to extend and maintain,
- Learning from the experiences of teams at other organizations, adapting their lessons into your own systems, and
- Ultimately, after more experience, designing your own systems.

This exercise also requires you to practice the skill of locating the further resources necessary to understand the design. The intial articles—blog posts, really—linked below are only starting points. You will need to decide what more information you need and where you might locate it.

The Articles

I have selected descriptions of four architectures and one failure.

Pick one, either one of the architectures or the failure story, as a starting point.

The architectures

Give meaning to 100 billion analytics events a day: Analytics pipeline at Teads, Part 1 (https://medium.com/teads-engineering/give-meaning-to-100-billion-analytics-events-a-day-d6ba09aa8f44), Part 2 (https://medium.com/teads-engineering/give-meaning-to-100-billion-events-a-day-part-ii-how-we-use-and-abuse-redshift-to-serve-our-data-bc23d2ed3e07)

> "In this article, we describe how we orchestrate Kafka, Dataflow and BigQuery together to ingest and transform a large stream of events. When adding scale and latency constraints, reconciling and reordering them becomes a challenge, here is how we tackle it."

Scaling Klaviyo's Event processing Pipeline with Stream Processing (https://klaviyo.tech/scaling-klaviyos-real-time-analytics-system-with-stream-processing-4b3bb87cd6b5)

> "In this article, I will cover the challenges of the initial version of Klaviyo's event aggregation system, the rationale behind choosing Flink as the streaming framework, and how we built and shipped Abacus."

Highly Available MySQL Clusters at WePay (https://wecode.wepay.com/posts/highly-available-mysql-clusters-at-wepay)

> "This post describes WePay's highly available MySQL architecture, and how we achieve short outage times during failures."

Netflix: What Happens When You Press Play? (http://highscalability.com/blog/2017/12/11/netflix-what-happens-when-you-press-play.html)

> "This article is a chapter from my new book Explain the Cloud Like I'm 10 (http://highscalability.com/blog/2017/12/18/explain-the-cloud-like-im-10.html). The first release was written specifically for cloud newbies. I've made some updates and added a few chapters—Netflix: What Happens When You Press Play? and What is Cloud Computing?—that level it up to a couple ticks past beginner. I think even fairly experienced people might get something out of it."
>
> **Note:** This article is written for a more basic audience than this class. I included it because many of us have used Netflix and may be interested in how the service works. Although written in simple terms, the content is accurate, as the author, Todd Hoff, maintains the HighScalability (http://highscalability.com/) Web site. You will need to follow up with readings at a deeper level of technical detail. The Netflix Technical Blog (https://medium.com/netflix-techblog) is a good starting point. Caution, though—there is *so much* material at the blog that you will have to limit how much time you spend there.

The tale of a failure

For those who just want to watch the world burn (https://www.youtube.com/watch?v=efHCdKb5UWc):

On Infrastructure at Scale: A Cascading Failure of Distributed Systems (https://medium.com/@daniel.p.woods/on-infrastructure-at-scale-a-cascading-failure-of-distributed-

systems-7cff2a3cd2df)

Distinguished Engineer Dan Woods (https://twitter.com/danveloper) summarizes a nasty failure at Target. Note that the failure is just the starting point—your goal in this case is the same as for the articles above, to analyze an architecture and its tradeoffs.

(Yes, I would have loved to include a report of the Mar. 13, 2019 Facebook/Instagram/WhatsApp outage but it looks like we won't have authoritative reports on its causes for several weeks.)

How to read the article

Here is a suggested process for reading the article. Your background and your article will vary, so these are only suggestions. The one point that I recommend most strongly is to first read for gist, not detail.

You will need to read the article several times, with breaks to search out and read other sources supplementing it. As with most technical articles, your first read should skip the details and scan for the main points. You should not assume that you can read it once and understand it.

Here are the sorts of questions to ask as you read:

1. What is the focus of this article?
2. What is the emphasis of the system design? Note that the article will likely focus on one part of the system design and you may need to track down other references to get the full picture.
3. What is the biggest challenge the designers faced? This could be inherently difficult tradeoffs such as response latency versus data consistency, tool tradeoffs such as limitations in the available software, combinations of these, or other things.
4. Are there apparent limitations to their design?

Practicality check

Early in the process is a good time to take a practicality check. Does this article and this project seem like a good focus for you? If you find it too vague, too far from your experience, too obscure, or otherwise unnecessarily difficult, consider switching to another. Do not do this too many times—at some point, you need to "pick 'n stick"—but also do not presume that your first choice forces you to commit.

Summarizing the article

After reading the article, try drawing a system diagram from memory. You may be tempted to return to the article to clarify fuzzy points. Resist that temptation, if possible, and try to complete your diagram without referring back to your souce. Put question marks where you simply have no recollection of the relationships.

Your diagram may highlight the same elements as a diagram in the article or it may differ.

In my experience, attempting such a diagram after first reading is humbling, highlighting how little I understood the first time through. Reread the article with an eye to filling in the gaps.

The article will likely mention unfamiliar tools such as Consul or Kafka. Note their names but do not follow up yet. Speculate on what their function is. (The Netflix article is the exception here, as it is written for a less technical audience. However, when you supplement it with more technical articles, you will experience this process.)

Refine your diagram to reflect your improved understanding. Repeat the process until you think you understand the relationships between the various pieces.

Identifying further sources

Now it's time to broaden your view. You may need to address several types of gaps:

1. More details about the specific system under description. How much is available varies with the sponsoring organization. Does the author or their group have a blog? What about the site for the product they describe? Go with what you can find. There will always be some parts left undescribed.
2. The purpose and design of component tools such as Kafka, Cassandra, or Flink. Go to the home site for the tool and read the overview. What other resources are on the site? Do a search comparing this tool with others in its class. There are often articles on Quora (https://www.quora.com/) of the form, "When would I use [X] versus [Y]"?
3. General background on underlying concepts. For topics in computing, Wikipedia often provides a good overview. Be alert for strongly partisan statements and unsupported opinion in the entriess, though.

With luck, the article will have links to related materials. You do not have to read every linked article but they do represent a good start towards completing your background.

Another good starting point is … the notes for this course (http://756.cmpt.sfu.ca/756-19-1/), both the daily class notes with their embedded links and the Resources page (http://756.cmpt.sfu.ca/756-19-1/resources.html).

What to read for in the supplements

Your are not trying to completely understand every detail. That would be an unending process. Instead, you are seeking to understand the design of the system described in the original article. A useful rule of thumb: When in doubt, do not follow up on a detail. If you really need it, that need will become clear when you go to write your analysis and need to justify some point. Most often, we spend too much time pursuing details (we're programmers —details matter!) at the expense of keeping our heads up and seeing the larger view.

As with any rule of thumb, apply this with moderation. Some details *will* be necessary to learn. This isn't an invitation to be vague, it's an reminder to keep your focus.

Begin writing your analysis *early*, when you still feel like you don't know enough. Tip: That "not knowing enough" feeling won't go away no matter how long you avoid setting fingers to keys. When you see a substantive gap in your understanding or logic, pause to find a source that resolves that uncertainty. Interweave the processes of reading and writing.

Once you have drafted your initial analysis, revise and polish.

> **Quality is more the product of multiple drafts than exhaustive detail.**

The report structure

Your report should have the following sections:

1. Introduction: What is the purpose of the system you are analyzing?
2. Components: What are the pieces from which it is built? Briefly describe them. The components include pre-built products such as Kafka or Cassandra, as well as parts written specifically for this product.
3. Build versus buy: Which parts of the system were taken off the shelf (they are independent tools that the designers acquired, whether open-source or for-fee) and which parts were purpose-built for this organization? Pay attention to the purpose-built components. They are a strong clue to a key goal of the system, one that could not be met using existing tools.
4. Architecture: How are the components organized? How do requests flow between components? Include a system diagram. Your diagram may be derived from one provided in your source documents but will need to be modified to suit your different purposes. Provide citations (including a hyperlink to the document) to any source diagrams.
5. Design goals: What are the design goals for this system? What did the designers consider most important? What did they consider simple to accomplish or unimportant? A design is as much defined by what it does *not* try to do as what it attempts. Define these goals using the terms set out in class. Refresh your memory with the class notes and the course learning outcomes (outcomes.html). Note: This semester, we will not meet every learning goal to the degree that I wrote on that page.
6. Tradeoffs: What did the designers have to give up or compromise in order to achieve their main goals? What goals were "nice to have" but "unable to do"?
7. Resilience to failure: What failures can the system accommodate? What failures might take it below its Service Level Objectives? You do not have to know the actual SLOs, as organizations will likely never publish internal targets. This question instead asks what failures would dramatically reduce the system's capacity to meet the business objectives of its designers.
8. Scalability: How readily can the system expand to support increased demand? Shrink when demand falls? Are there any bottlenecks?
9. Next steps: What should the designers focus on improving next? What might have to change if user demand doubled? Do any parts seem brittle and unncessarily prone to failure?
10. Conclusion: Summarize the design, restating its main design goals, components, and tradeoffs. What parts impressed you? What parts do you think could have been done better?
11. Bibliography: List your references, both those directly cited and those that you reviewed but did not cite. I do not require any specific format, especially given that for a project such as this every reference is likely to be an on-line source. Do provide a title, the author (where applicable), and a URL (if any).

Citations: This report is not an academic paper but the sort of report that you might prepare for internal use at an organization. As such, you can be less formal in citations. The only times I want a direct citation are:

- A quotation
- A diagram that was the basis of one you drew (when in doubt, cite)
- Empirical results such as measured performance statistics

Otherwise, simply list all the articles that you consuled in your bibliography.

Submission

Length: Your report should be between 10 and 20 pages.

Submit your report as a PDF to CourSys (https://coursys.sfu.ca/2019sp-cmpt-756-g1/+final-proj).

Grading Rubric

1. Context (5%)

- Describes purpose of system
- Presents external goals that determine design (for example, automated financial trading requires extremely low latencies)

2. Argument (20%)

- Structure of the argument is clearly shown
    - Broken into sections
    - Appropriate headers for each section
    - Transitions connect the previous section to the new section, in light of the overall argument
- Detailed logic
    - Progress from established points (old information) to next point (new information)
    - Mechanisms presented in sufficient detail to justify claims (for example, how does a given feature reduce reliability or increase latency?)
- Concepts and terms defined at point of introduction

3. Architecture description (30%)

- System diagram
    - Parts clearly labelled
    - Demonstrates the points made in your article
    - Minimal amount of irrelevant detail (using “borrowed” diagrams is fine but if possible, edit out irrelevant detail—see below for example)
- Information flow shown
- Components named and their role described
- Rationale provided for any purpose-built components rather than off-the-shelf solutions

4. Design (30%)

- Design properties described in terms of appropriate metrics and invariant guarantees (Week3-Mon.html)
- Key design goal(s) identified
- Driving criteria for the design
- Tradeoffs described
- Important components or aspects that the designers did not describe are noted
- Failure resilience described, backed up by sufficient detail
- Scalability assessed, backed up by sufficient detail

5. Evaluation and next steps (10%)

- Evaluation focused on key design criteria
- Realistic expectations—acknowledges that all systems have gaps or lacks
- Next steps potentially contribute to the system’s main purpose

6. Overall (5%)

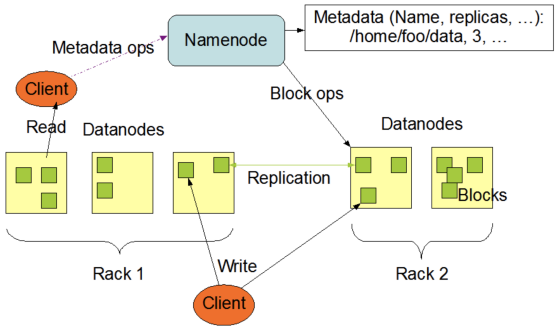
- Logic flows

- Balanced presentation

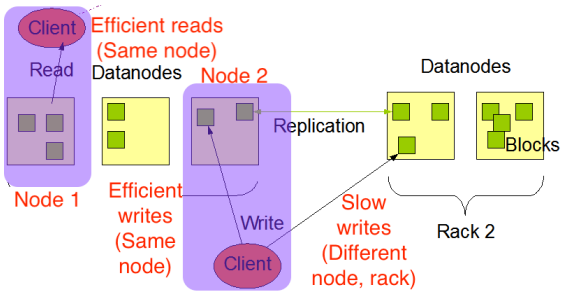
Frequently-Made Suggestions

Here are some suggestions that I've made several times in individual student consultations.

1. Do not include the prompt questions in your final draft. "Prompt questions" are the questions I listed under the report structure to prompt your thinking.
2. Make your structure clear by including header sections.
3. Use the suggested structure as a starting point but as your argument develops, feel free to develop your own sequence. Cover the points I request but do it in whatever order makes the case most clear.
4. If you copy phrases or sentences you must put them in quotation marks or a block quote. Cite the source.
5. If you copy a diagram, cite your source.
6. Editing a diagram from a primary source: Often a primary source has a diagram that captures much of what you want but has both extra detail irrelevant to your point and missing detail. In such cases, I often crop the diagram and add my own annotations. For example, here is a diagram from the HDFS Architecture Guide (https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html):



   And here is what it looked like after cropping to the part of interest, adding text (in red), erasing some lines, and adding node markers (lavender rectangles):



7. If the designers use an off-the-shelf component that has well-described tradeoffs and that component is central to their system's performance, the component's tradeoffs become part of the system design. For example, if their primary data store is a database that emphasizes availability and throughput at the expense of consistency (for example, supporting "eventual consistency"), then a system that relies on that component is also making that tradeoff.
8. If the designers do not cover a topic or describe all their tradeoffs, there may not be much you can say about that issue. Focus on the design decisions and properties that are described. Companies will often limit their descriptions to protect trade secrets or simply from lack of space in their article.