416 Distributed Systems [open ended]

Multiple deadlines (see below)

Winter 2022

fddc792

This is an open-ended project that must be completed in a team of 5-6 people. **Optionally** you can also, (at least partially), deploy it on Azure. The extra number of people (over A3) will provide you with developer power to execute on an ambitious project.

Type of project

Your project must address a non-trivial problem related to **distributed systems**. It must include a substantial software effort in Go. Note that 'substantial' includes complexity and not just code size. The most direct way to satisfy the project requirement is to prototype a distributed system design. Such a system can be built from scratch, but the project can also be formulated as a non-trivial extension to an existing system. The idea behind the system does not need to be original. The majority of the distributed logic in the implemented system must be implemented by the project team.

As a benchmark, your project must have at least the complexity/difficulty of assignment 3.

Project constraints (evolving):

- Go must be used for the core distributed logic in the system. However, other languages may also be used in the project. For example, you can build a distributed system in Go and have Android clients, implemented in Java, that connect to it and use it.
- The system must be able to support node churn: nodes that fail and leave the system, as well as nodes that join the system.
- The system cannot be embarrassingly parallel: there must be some distributed state and coordination between nodes in your system.
- **Optional:** Some part of the system deployed on

Azure.
- The system must be well tested.

Project ideas

Here are several project ideas. Treat these as inspiration; I strongly encourage you to come up with your own project idea. If you have an idea and you would like to discuss it, consider posting a description to Piazza.

Project idea: Consensus-Based Failure Detection

After A2 you've become more interested in failure detectors. You start reading more and realize that the failure detector you developed in A1 is not robust to network partitions, which may cause inconsistencies in the system about the state of a node. For example, Node A and Node B are monitoring Node C. If Node A is partitioned from Node B and Node C, then Node A will incorrectly report that Node C has failed while according to Node B, Node C is still alive. You find out about the Paxos algorithm and realize that this problem is in fact a consensus problem. You decide to implement a consensus protocol, like Paxos or Raft, on top of your failure detector library to decide if a majority of systems believe that a node has really failed.

Project idea: Regionally Restricted Streaming Service

If you were to take a random survey of current UBC students to determine the answer to the question "How do you procrastinate?", the majority of the responses would have something to do with watching Netflix. But as a current 416 student, you have already been exposed to web-proxies and CDNs. So, now your favorite way of procrastinating is to build Netflix from scratch instead of watching it. To do this, you would need to make a video streaming service built on top of a custom CDN, which provides regional restrictions. To make your system more robust, you may also choose to use a distributed key-value store to house your user data.

Project idea: A version of DynamoDB

"Amazon is expanding in downtown!" is something you have gotten used to hearing after living in Vancouver the

last few years. As an upper-year CS major, you have started looking for well-paid full time jobs and identified Amazon as a viable destination. Being a smart group of 416 students, you decide to build Amazon's DynamoDB from scratch as you think that would be a good way to impress an Amazon recruiter. You also like how they use CRDTs to provide eventually consistent reads but you are also interested in finding out how they can also provide strongly consistent reads across their whole distributed database. And, to widen your scope, you decide to use your newly built DynamoDB on Azure instead of AWS to increase your chances of impressing a Microsoft recruiter, as well!

Project idea: Build an anonymity network

Tor is an anonymity system built on onion routing. Tor allows clients to obfuscate their network identity/location (IP address). The idea is simple, but supporting multiple clients, defending against attacks, and providing good performance to clients (e.g., responsive browsing) are non-trivial requirements.

One version of this project is to prototype a basic version of Tor, and **optionally** deploying it on Azure, and demonstrating that you can use it to browser the internet. A basic version might include:

- Handling connecting/disconnecting guard/relay/exit nodes
- Secure onion routing (intermediate hops do not observe payload)
- Circuit setup/tear-down protocols
- Periodic circuit refresh to avoid using a circuit for too long

Tor is just one type of anonymity system. If you are interested in this space, there are a variety of other system designs that you can adopt (e.g., Vuvuzela). Or, feel free to create a new one!

Project idea: Build a peer-to-peer machine learning system

Machine learning is all the rage. There are many distributed frameworks, but all of them assume a centralized learning process with access to a central store of training data. Build a peer-to-peer solution for learning a

global model (of a variety of your choice) that has as few centralized components as possible and where data is spread across peers. Assume an adversarial context in which peers do not want to reveal their data to others. For this project you may want to recruit to your team someone who has taken CPSC 340 (and has done well in it). You can also substantially expand the security/privacy requirements of this project. Take a look at the Biscotti paper for an example of a sophisticated system in this space.

Project idea: Build a distributed web crawler/search engine

Web crawling is kind of a 90s topic. But, an efficient and scalable version is a complex distributed system with many interesting pieces. An assignment from 416-2016w2 describes an 'assignment' version of a web crawler that is a good starting point. This version described a set of worker crawlers that are spread over multiple data-centers, a web-graph that is maintained in a distributed fashion, a distributed page rank computation, and keyword search capability. You could extend this version or consider building a different variant.

Other project ideas

- Build a fault-tolerant parallel computing platform based on Spark's RDD abstraction.
- Build a peer-to-peer version of DropBox based on the design of XFS.
- Build a distributed object system, like Emerald, but without a compiler.
- Build a distributed shared memory system, like Treadmarks.
- Build a distributed assertions mechanism for Go systems that can check a distributed system's properties at runtime, using Dinv as inspiration.
- Implement a byzantine fault tolerance algorithm, a classic example is PBFT.

Project structure

Each project group will be assigned to a TA. This TA will be the point person for project advice, deliverables checking, weekly meetings, and other project-related logistics. You will do all your work in a git repository hosted by UBC

enterprise github (you can make it public later, if you wish).

The required project deliverables are listed below. In cases where the deliverable is a written paper, I would prefer that you share the doc with the TAs and Ivan as an editable google doc. If you would rather use another submission approach, let me know. I would prefer final project reports in pdf format for an ACM SIG of your choice.

- **Project proposal draft:** a paper (about 5 pages) describing as much of the project proposal as possible. However, the draft does not need to have well-defined milestones.
- **Project proposal:** a paper (about 5 pages) detailing the problem you plan to address with your distributed system, your proposed approach, and a realistic timeline for your team's efforts. See proposal details below for more information. The proposal must detail three well-defined milestones. Each milestone must include (1) deliverables that you will share with TAs+Ivan for the milestone, (2) a written document that explains the deliverables and their status. The best way to think of the milestones is as a contract: if I accept your proposal and you meet the milestone you describe, then you will receive the full mark for the milestone.
- **1st project milestone:** 1st milestone deliverables and a document that explains the deliverables and their status. For example, if the deliverable is code, then the document must describe what the code does, under what conditions it works, how to run and deploy the code, etc. If the deliverable is a dataset, then the document must explain the format of the data, what it means, how it was collected, how complete it is, etc.
- **Project update meeting 1:** Your project group will meet with a TA assigned to your group (and maybe Ivan) to discuss your project status. You will discuss the first milestone, your progress towards the second milestone, and any outstanding questions/concerns that you might have.
- **2nd project milestone:** 2nd milestone deliverables and a document that explains the deliverables and their status.
- **Project update meeting 2 (trouble groups only):** If you have done poorly on your first milestone and we decide that your team is struggling, we will meet with your team again after the second milestone.
- **3rd project milestone:** 3rd milestone deliverables and a document that explains the deliverables and their status.

- **Project demo:** a live demo of your project work to TAs and Ivan during finals week. During the demo we will ask your team questions about your system implementation and design.
- **Prototype implementation:** git repository with your code.
- **Project report:** a paper detailing the problem you set out to solve, design of your system, implementation description, and some evaluation results. The final report (due at the end of the term) should be no longer than 8 pages (excluding references) and should resemble a research paper. See final report instructions below for more information.

Through the course of your project work, it is strongly recommended to have **weekly** meetings with your assigned TA for the project during his/her office hours. The whole team need not show up, the team lead is enough to provide your TA your team's status update, discuss concerns, queries or get feedback etc.

Proposal

A project proposal details the problem, your proposed approach/solution, and a realistic timeline for your team. The proposal must include at least the following sections: introduction/motivation, background, proposed approach/solution, evaluation methodology, timeline.

You should aim for a proposal that is about 5 pages long. Shorter and you're probably missing some detail; longer and it becomes too detailed and too long to read. That said, there are **no** page limits (lower bounds nor upper bounds) on your proposal. Note that a proposal draft is a proposal without milestones. All the items in this section apply to proposal drafts.

Here are two high-level ways in which I think about your proposal:

- **A proposal is a contract.** If you build the thing described in the proposal then you get a perfect mark on the project. But, writing good contracts is hard work. For example, a good contract must be precise (it should be clear what you are and are not going to do).
- **A proposal is your opportunity to convince me that you know what you're getting yourself into.** I won't let you do a project if I know that you do not stand a reasonable chance of succeeding at it (this is

a distributed system course, not an SE course :-) So, the proposal should convince me that you know what you're doing -- that you've thought about the key issues (you know what they are, approximately how you're going to solve them), you know what resources you will need/where you will get them (technology/libraries/algorithms/data sources/hardware/etc), that you thought about how to manage your time and how to manage the team roles and responsibilities (who does what/when), and that it all adds up to a realistic plan for a successful project.

Here are three example proposals from an earlier instance of this course (include SWOT analysis, which you do not need to include; do not include milestones):

- Heterogeneous Dynamic BSP programming in Go with a Pregel-inspired API
- Live Pod Migration in Kubernetes
- Distributed Key-Value Store Utilizing CRDT to Guarantee Eventual Consistency

Here are three example proposals from a graduate course (these include milestones):

- SyWoWa: a System & Workload Aware Graph Partitioner
- CRDTree
- Distributed State in PGo

Detailed proposal instructions.

- The timeline must include dates and milestones/deliverables. It must be sufficiently refined to include milestones that are specific to your project. Do not simply list the deliverables without listing the internal project deadlines. The timeline is there to get you to think about your time and to loosely commit to a schedule.
- This is a distributed systems course, so make sure that your proposal is focused on issues/challenges/objectives relevant to this topic. If you can, try to focus on distributed abstractions: which ones will you be using, developing, and how will you evaluate their qualities.
- The bulk of your proposal must be dedicated to design: what will your system look like, what properties will it have, what features will it include/omit, how will clients interact with your system, etc. This is the most important section.

Writing this section well is difficult; spend the time to do a good job on it. The best way to write this design section is to look at A1, A2, A3 specs and model your design description based on those pages.

- It is important to omit content that is irrelevant to your proposal. Before including text, consider whether or not it plays a purpose in explaining your proposed system and its objectives. If not, then it can probably be cut.
- Consider giving your project/system a name. This way you can easily refer to it in your proposal.
- Your project can re-use external code/algorithms/ideas that you find online (e.g., open-sourced Paxos Go implementations). Leverage prior work and build on it to avoid re-inventing the wheel and to get to interesting ideas quicker (e.g., implementing Paxos is itself a complete project).
- Your proposals may end up including highly specialized content (e.g., details of crypto algorithms). Make sure to define non-standard/specialized terms, include examples, and intuition -- anything to help get your ideas across. This work will pay off in the long term: (1) it will get you thinking more deeply about your work, and (2) you can re-use it in the project write-up.
- Make sure there is logical flow to your proposal. Define terms before you use them, motivate particular perspectives before launching into details, discuss existing systems or previous academic work necessary to understand your proposal before you rely on it for your descriptions. You do not have to provide an academic treatment of related work, but it does not hurt to read a bit about your topic and include references to inform your content.
- You build your system so that you can eventually deploy it and run it. Your proposal must include a section on *evaluation methodology* in which you explain how you will evaluate that your system works as expected. For example, you might optionally also deploy your anonymity system on Azure VMs across different data centers and measure the end-to-end throughput of your system. Evaluation methodology should match your system goals; e.g., if your system provides access to a resource to many clients, then you should evaluate your system with many clients.
- Consider including info-graphics/figures to explain your design. Sometimes it is easier to explain a complex idea with a picture (consider diagrams in the A3 spec). Likewise, don't hesitate to include formalism/math to explain your ideas (though, be

careful with including formalism for formalism sake -- make sure it helps to explain rather than confuse the topic).
- A well thought-out and detailed proposal will only benefit your group in the long run -- you will have a more clear idea of what you are really working on!

Submitting you project proposal (draft):

- **[Draft proposal submission]** To submit a project proposal draft, create a private piazza post. For this post use the title: "Project proposal draft: [[title]], list of CWLs for your team" with with [[title]] replaced with your project title/name. The easiest way for us to give you feedback is if your post includes a link to an **editable** google doc containing your proposal. Make sure to identify the group members in the pizza post and the proposal body.
- **[Final proposal submission]** If you submitted a google doc that we can access (above), then continue working on that doc and we will snapshot your google doc at deadline time. If you did not submit a google doc, then you can update your piazza post with the final proposal doc copy, preferably in pdf format.

Your proposed project might evolve

The proposal is your best effort at scoping out the challenges that you expect to come up against and some ideas/plan on how you will resolve these. But, of course, system design and software engineering is not that predictable.

It is difficult to describe how much you can deviate from the proposal. So, UDP instead of TCP may not be a significant change for some proposals, but could be a major change for others (e.g., if you are investigating distributed congestion control adaptation in TCP and now change to UDP, the difference is major!).

Please discuss potential major changes with the TA assigned to your group and/or with Ivan.

Prototype implementation

There are no constraints on your distributed system design

and implementation outside of the ones listed at the top.

- We will create a survey where you can specify your team members. We will then create a repository with the name **Project-[CWLs-list]** using the CWLs of the team-members on your team.
- We will read your code.
- We will read the code that is in your repo by the deadline of the respective deliverable.
- Note that this version of the code **must correspond to what you describe in your milestone/report**. For example, if you describe a Tor-based system in your report and do not note any work-in-progress items, and we do not see any onion encryption code in your repo, your mark will be penalized.
- It is okay if your code does not work completely! Your report should note what currently works and what doesn't work.
- No, you are not required to include code comments, compilation instructions, or anything else that would make our code-reading lives easier. (Though we would certainly appreciate any such effort).

Report

Your final report is a description of the problem you attempted to solve, what you have built to solve the problem, why you built your system the way you did, and how the system works/doesn't work.

Detailed report instructions.

- Report must be 8 pages max. This includes all the things that you want Ivan/TAs to read/see, including all diagrams.
- Use your group's github project repository to submit your report as a **pdf document**. Place your report into `report/report.pdf` at the top level of your repository (if you use LaTex, make sure that it is compiled into a pdf).
- Use whatever format you want, but please don't torture us with font size 8 and awful margins. I recommend the 2-column ACM article format.
- You can copy/paste and reuse text from the proposal.
- But, of course, don't plagiarize other's work! Attribute all the images/text you borrow; standard writing practices apply.
- Your report must stand on its own -- cannot refer to proposal or to a youtube video where you explain

your system in an hour-long lecture.
- Any evaluation results must have a proper methodology to introduce the results. What was the goal of the evaluation? Why did you measure what you measured? Typically, the more information you provide to describe your experiments, the better. But, it requires careful judgment to report just the important details.
- The report must describe the system whose code you are submitting by the code/report deadline. This means that if you have some bits that are unfinished, but you plan to finish them for the demo, then you must **explicitly note** in your report that they are a **work in progress**. Note that the ShiViz extra credit cannot be a work in progress item; we have to see ShiViz diagrams for your system in the report.
- The report should include an approximate description of your demo script. The demo has specific requirements (see below), and we want to see an outline of your demo plan that matches these requirements in your report. This doesn't have to be long: a short paragraph per demo stage is fine.

Project demo

The project demo is a 45-minute adventure. You will demo your project to Ivan and a group of TAs in private, including a technical Q/A regarding the project design and implementation.

Detailed demo instructions.

- The github project repositories will **not** be frozen after you submit your code and report. So, you can continue to use your repository to develop and improve your system for the demo! Yes, that means that you can add new code/change existing code/etc.
- No, we don't care how much new code you add between report and the demo -- if only 10% of your proposed system is built by report-time (and 90% is a work-in-progress), then expect penalties on the code/report. The demo is a separate beast marking-wise.
- If you are working on the EC, you must generated GoVector logs and use ShiViz live during your demo to receive full EC marks. You can do so during the normal operation step, or another step in the demo (below).

- Your demo is **45** minutes long. Here are the components/time/demo-mark break down:
    - Demonstrate *normal operation* of your system (no failures/joins) with at least 3 nodes.
        - 15min expected
        - 40% of demo mark
    - Demonstrate system can survive at least 3 node failures
        - 10min expected
        - 20% of demo mark
    - Demonstrate system can *join and utilize* at least 3 new nodes
        - 10min expected
        - 20% of demo mark
    - Design Q/A
        - 10min required (we will stop you at the 35min mark to do Q/A)
        - 20% of demo mark
        - We will ask questions of the entire group and anyone on your team can answer.
- There are several critical notions in the rubric above that will vary from system to system (group to group):
    - Normal operation: show that your system achieves its stated function (e.g., serves HTML to web-browser clients from a CDN, sends email via ToR, etc)
    - Survive: show that your system's normal operation is not disrupted by the failures (e.g., game continues to be playable after failures)
    - Utilize: show that your system actively uses the newly joined nodes (e.g., database integrates and uses new nodes to store keys/values)
- To get full marks on the demo you must (1) define the above in the report or in the demo, and (2) demonstrate to us that the above conditions are satisfied by your system during the demo (e.g., when you fail/join nodes). You can do this by some of the following:
    - Show us terminal output with copious verbal explanations
    - Show us a web browser GUI that shows us blinking lights that semantically match the above goal
    - Robots that behave as expected (where you defined for us what is expected)
    - Some other means (typically runtime system I/O)
- For failures, you can decide which nodes to fail and how (though if you fail the Azure LB that has a standby that you did not build.. you won't be getting

much/any of that 20% *survival* mark).
- Yes, we want to see you inject failures, preferably on a terminal with a Ctl-C signal. Same for node joins.
- In case your project makes use of Azure in some way, you have to explain/show that this is indeed the case.
- Your system must use a real network between your nodes -- distributed systems that uses localhost for communication will be severely penalized.
- Note that your demo slot is tight -- we may have scheduled other groups before/after your group. I strongly encourage you to practice your demo multiple times and develop a robust demo script. It helps to curate the demo env and set it up just the way you want it.
- Some projects may have special requirements (e.g., prohibit failure of 3 nodes). If this is the case, post on piazza to arrange a change to your demo components. You must discuss these with us before your demo and we have to sign off on any deviations from the above in writing via a piazza post.

Deadlines

All project 2 deliverables are due at 11:59PM on their respective dates. The project is structured as a series of regularly occurring deadlines Do not miss these!

- March 18 : Project proposal drafts
- March 25 : Final project proposals
- April 1 : Milestone 1
- April 4-8 : Project update meetings
- April 8 : Milestone 2
- April 15 : Milestone 3
- April 19 : Project code and final reports
- April 18 - 22 : Project demos

Grading scheme

The project is 45% of your final mark. Here is the mark breakdown across the different deliverables:

Proposal draft	2%
Proposal	5%
1st Milestone	7%
2nd Milestone	7%
3rd Milestone	7%
Demo	7%

Report and implementation	10%
Total (of final mark)	**45%**

Extra credit

This project is extensible with one extra credit option.

- EC1 [1% of final mark]: Add sufficient tracing to your system to be able to observe the normal operation execution of your system across **all** the nodes taking part in the execution using ShiViz. The corresponding ShiViz diagrams must match your system design and should help explain how your system is implemented.

  The diagrams and explanations of the diagrams must be in your final report (aim for 1-2 diagrams in total). Please store the logs for your diagrams in the report in your repository. You must also show us a live demo: generate trace output during the normal operation part of the demo, visualize the resulting logs with ShiViz, and explain the resulting diagram to us.

Make sure to follow the course collaboration policy.

Last updated: April 20, 2022