CS425, Distributed Systems: Fall 2022
Machine Programming 4: IDunno, a Distributed Learning Cluster

Released Date: Nov 9, 2022
Due Date (Hard Deadline): Sunday, Dec 4, 2022 (Code+Report due at 11.59 PM)
Demos on Monday Dec 5, 2022

Clarifications made Nov 14 2 pm US Central. Please see red font below.

The is a very intense and time-consuming MP! So please start early! Start now!
(Disclaimer: Like all MPs and HWs, all references to all companies and people in this spec are purely fictitious and are intended to bear no resemblance to any persons or companies, living or dead.)

ExSpace (MP3) miraculously merged with two social media companies Qwitter and KitKat to form a new conglomerate called…MartWall Inc. (Sidebar: if you have not seen the movie "Idiocracy", Indy highly recommends it. Relevant to this MP's joke is the "Costco" scene here: https://www.youtube.com/watch?v=sdNmOOq6T8Y . Anyway,…).

MartWall loved your previous work at ExSpace (and they're also aware of your great work on the mission to Mars in HW3), so they've hired you as a "MartWall Fellow". That's quite prestigious! Congratulations!

You must work in groups of two for this MP. Please stick with the groups you formed for MP1. (see end of document for expectations from group members.)

Side note about this MP. This MP requires you to learn the basics of systems support for Machine Learning (ML) by yourself (without needing to understand internal details of how ML models work. For practical purposes, the information we provide in this MP spec doc should suffice (in terms of basics), but you can feel free to look up online resources for more basics in ML. Treat this MP as a way for you to use your familiar knowledge (from CS425), as you learn an unfamiliar area (ML), and where you build a system that combines both familiar and unfamiliar unknowledge. This MP will take you well outside your comfort zone, and that is ok!

Now, back to the MP. MartWall needs to fight off competition from their two biggest competitors: Pied Piper Inc., and Hooli Inc. They also have to fight the scourge of fake news. To do this they have decided to use…Machine learning, and in particular Deep Neural Networks (DNNs)!

Your job is to help them build a simple Machine Learning (ML) platform that does both **training** and **inference (aka "model serving" or "prediction")**. Your system: a) does

training (you will be able to skip this stage by using pre-trained models), and b) does agile inference where any model and query can be specified, by reading queries from SDFS and writing results to it. You can read online what these terms mean in Machine learning. (Learning new skills online in a fast-moving Distributed Systems world, is an important skill!)

You can reuse any and all code you like from open source, especially TensorFlow (an open source machine learning platform released by Google: https://www.tensorflow.org/), PyTorch (https://pytorch.org/) or Spark (https://spark.apache.org/). Relevant is the Ray system (https://www.ray.io/).
Make wise choices! Please note that these different platforms have different memory usages (which might hinder which models you might be able to deploy given the memory limits), so we recommend you try some of the other systems for their memory usage before you pick one.

Your machine learning system is named "Illinois DNNs", or in short, "**IDunno**".

These tasks below are sequential, so please start early, and plan your progress with the deadline in mind. Please DO NOT start a week or two before the deadline – at that point you're already too late and will likely not be able to finish in time.

Machine Learning Basics: A Deep Neural Network (aka a "model") is "trained" on a dataset (called training set), and then does inference (or prediction) on other data not in the training dataset. The training phase is time-consuming, and it trains the "parameters" inside the neural network graph. Subsequently, the inference (a.k.a. prediction) phase outputs a "decision" for each test sample. For instance, in the training phase a DNN may be trained on a variety of images that contain cats or not, with each image tagged as containing a cat or not (this is called "supervised learning"). This trains the DNN to recognize cats. Now in the inference phase the DNN can be given another new set of images and needs to predict whether each contains a cat or not. (This application is called "Image Classification").

Notice that the training set and the testing (for inference) set must be disjoint! If they are the same, then the DNNs will perform perfectly, i.e., it will be accurate all of the time. It is only when training dataset and testing dataset are disjoint that the accuracy is really tested for the model you trained.

For this MP, you can use different systems for training and for inference.

IDunno Specification: Below we provide a sketch of the expectations from your IDunno system. Like all software projects, the description is incomplete, and it is up to you to fill in the rest of the details, both design and implementation.

IDunno is intended to support training and agile inference for any two neural networks – from among Resnet-50, Inception V3 with ImageNet dataset (for image processing), and NMT (or GNMT) for neural machine translation (speech). Here are some starter links for these DNN models (feel free to find other links on the Web). You are not bound to use only these; you can feel free to use other implementations of the above-listed models.

- Alexnet (typically fits within 2 GB)
  - https://github.com/pytorch/vision/blob/main/torchvision/models/alexnet.py [Wikipedia: https://en.wikipedia.org/wiki/AlexNet]
- MNIST Dataset with LeNet model
  - https://en.wikipedia.org/wiki/LeNet
  - https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-from-scratch-for-mnist-handwritten-digit-classification/
  - https://pyimagesearch.com/2016/08/01/lenet-convolutional-neural-network-in-python/
- Resnet-50 or even Resnet-18
  - https://github.com/keras-team/keras-applications/blob/master/keras_applications/resnet50.py
- Inception V3
  - https://paperswithcode.com/method/inception-v3
- NMT
  - https://nlp.stanford.edu/projects/nmt/
- You can use other models that are comparable in complexity and size as the ones above (please don't select simplistic or trivial models). Be wise in your choices.
- Your system should have the ability to support any two neural nets/models, but for the report and demo you can pick which two models you work with (that is, we won't test for generality across models – this is just to make your life easier).

(Note: By default you should process one query at a time per VM. If you batch queries, select your "batch size", i.e., number of inputs processed simultaneously, low enough, so that your inference fits within the memory of a single CPU.)

But IDunno must also generalize to run other types of neural networks, i.e., you can't just implement for two of the above DNNs and be done!

For this MP you can reuse code from the Web for Ray, Keras, TensorFlow, Resnet, MNIST, Alexnet, Inception-V3 and NMT (or GNMT), etc., however there should be a base IDunno system that you have designed and implemented by yourself.

First, IDunno trains the chosen DNN models (using training sets), then runs inference on it.

During training and inference, the datasets must be stored in SDFS (MP3), and read and written from/to it. All other data and files (e.g., exported parameters) must also be stored in SDFS. Please do not use the native file system for storing anything!

The different phases of IDunno are as follows:

1. A **Training Phase** that trains a particular DNN on a particular dataset. Note that when IDunno is in the Training Phase, no inference jobs are being run, but multiple training jobs can be run. For this MP the Training phase can just fetch and initialize the models for the next phase.
2. Once you have trained models, IDunno can be started in the **Inference Phase**. All inputs are in the SDFS (in files), and outputs from inference should be written to SDFS as well. No training runs during this time, but inference may run on multiple models! In the inference phase, queries come in for inference. When a query comes in, IDunno uses the appropriate DNN (model) to do the inference on. The main metrics are accuracy (which is model- and data-dependent) and query rate ~~inference time~~ (which is in your control). For the scheduling in IDunno, you should **not reuse** Ray's code (but you can look at Ray's code and design docs, and feel free to borrow some of their design choices). (We will check your code against Ray's code using MOSS). Mainly think: what are the key, most minimal, most important decisions you absolutely need? (Ray has a lot of other extra code and implementation that you don't need.)
3. The above training phase and inference phases do not overlap in time. Implement commands to start and stop each phase (for the training phase, it can just load the pretrained models). Also implement commands to add and remove a job, for each phase.
4. You can assume at most 2 models are running simultaneously. (Extra optional challenge: arbitrary number of models).

In this MP4 (IDunno), you have **two** challenges:

1. **Challenge 0: Fair-Resource Training**: You can use pre-trained models from the Web. You don't need to implement training by yourself as the VM resources may be insufficient for this. Please ensure that your "testing set" for Challenge 1 (Inference) below is different from the data used to train the model (for each model).
2. **Challenge 1: Fair-Time Inference**: Each model receives queries at a fixed rate – essentially each instance of the model fetches one (or a fixed size batch of) query at a time, and processes it, and then fetches the next query (or batch). Multiple copies of a given model should be run on different VMs so that they can process inference queries parallelly. Use all the VM resources, i.e., don't under-utilize resources. Implement client-side commands to specify hyperparameters like batch size ~~the query rate~~ for each model. (The TA will specify these hyperparameters ~~query rates~~ at the start of the demo). IDunno of course does not know a priori what these rates are. Your main challenge: For scenarios where several jobs have been added, and no more jobs arrive or leave, can you adjust dynamically the resources (VMs) among the running jobs so that ~~**their average inference times per query**~~ **the query processing rates of different jobs are within 20% of each other (10% of each other if possible)?** Because different models may take different times to

process queries, a model whose queries take longer may need more resources to match the processing times (within 20% or 10%, whichever you prefer). Note that once you have set hyperparameters (like batch size) you should not change them (i.e., the only thing that should affect the query rate is the amount of resources your IDunno scheduler gives to each job). Finally, note that when you have multiple VMs for a model (job), you have to ensure that these different VMs are not repeating each others' work, but are each fetching a different (sharded) set of queries. That is, doubling the number of VMs should result approximately in doubling of the query rate (for a given job).

3. (For the above items, you will find that Ray does some or most of this. You can browse their code. But your goal is to write a "faster", "leaner" version of what Ray has, given your workload goals (see later).
4. Please ensure that you your testing (query/inference) set is different from the training set, but that they both "belong together". E.g., if you use a completely different testing set (that is not paired with the training set), your inference may not work.
5. Note that you are doing inference on CPUs, while ML models are often run on GPUs. While most GPU ML models also run on CPUs, YMMV – be careful and pick ones that do.
6. **Challenge 2: Fault-tolerance**: See below.

Challenge 2: Fault-tolerance: IDunno must be fault-tolerant. Nodes may crash and never recover (fail-stop model), but the system should also handle failure recovery. When the failure occurs, it must be detected (use MP2 for this), and then parts of the job must be restarted quickly. Any "lost" queries must be handled and recomputed appropriately to get the same final result. Think carefully about the mechanisms you use! More information on fault-tolerance appears later in this document. You can assume at most 3 failures at any time before the system converges/quiesces.

Designate one of your VMs/machines as a coordinator, another machine as a client, and use a third machine as a hot standby coordinator. Use the remaining VMs as worker machines. The coordinator is used to submit and track jobs.

The coordinator may of course fail (you can assume only 1 coordinator fails at a time). If the coordinator fails, the hot standby should start kick in **quickly**. You can assume no other VM fails between a coordinator failing and it being replaced. While the coordinator is failed or being replaced, the ongoing jobs must **not stop processing data, and they must continue doing so correctly, i.e., without further data loss**. (Recall that SDFS writes and reads should work in spite of a failure). The only effect of coordinator being down is that new jobs cannot be submitted (until a new coordinator is available). It is ok to limit the IDunno cluster to accept one job at a time, adjust, and then admit the next job, but of course the cluster may have up to two jobs running simultaneously.

Other than the single coordinator failure assumption, your system should be tolerant to up to **3 simultaneous machine failures** (when the coordinator is down you can assume zero further workers fail until a new coordinator comes online quickly). Note that this implies that when workers fail, the coordinator does not. When a machine fails, the coordinator must restart the parts of the job on the remaining workers. Your first goal should be to hide the failure's effect and restart parts of the job **automatically** (and not manually). Your second goal should be to rebalance the resources across the different jobs (since failures may affect one job more than the other) to continue ensuring fair-time inference (even if the failed machines do not rejoin). Also if a worker rejoins the system, the coordinator must consider it for new tasks, and obey fair-time inference. The end result of processing a dataset must be the same, regardless of joins, failures, and leaves of coordinators and workers (within the above parameters).

In Inference phase, no matter the failures, **no queries should be lost by IDunno**.

Logistics: While DNNs typically run on GPUs, it is ok to run IDunno on CPUs provided in the VMs. This will be slower, but it is ok for this MP.

Use the code for MP1-3 in building the IDunno system. Use MP1 for debugging and querying logs, MP2 to detect failures, and MP3 to store the data (for inference, and the results).

As usual, don't overcomplicate your design. MartWall Inc. is watching, and if they find you've overcomplicated things, they will throw you into their supermarket (or throw you against the wall, whatever that means).

Make sure you design a reasonable UI (command line is ok) to start and stop jobs, see streaming results (e.g., dashboard, etc.) These will be useful during the demo.

We also recommend (but don't require) writing tests for basic scheduling operations. In any case, the next section tests some of the workings of your implementation.

These tasks are sequential, so please start early, and plan your progress with the deadline in mind. Please (please!) DO NOT start a week or two weeks before the deadline – at that point you're already too late! (It is also possible that some of your Fall break will be needed for doing this MP. Though, if you plan well, you might be able to do it without working on the break. Plan well!)

Create logs at each server (so that they are queriable via MP1). You can make your logs as verbose as you want them (for debugging purposes), but at the least a worker must log each time a IDunno task is started locally, and the coordinator must log whenever a job is received, each time a IDunno worker task is scheduled or completed, and when the job is completed. Make sure you use unique names/IDs for all workers, servers, etc. We will request to see the log entries at demo time, via the MP1's querier.

Applications: Write **at least two** applications using IDunno: choose from among Alexnet, MNIST, Resnet (any version), Inception-V3 with ImageNet, NMT (with one of the many language datasets available), or any reasonably well-known DNN model. Try to use datasets that are at least 100s of MBs large (if possible, even larger). For inference, run at least several minutes of inference workload.

How high can you push the queries per second, without affecting latency? In other words, draw the latency vs. throughput tradeoff curve (latency on one axis, throughput on the other axis).

Machines: We will be using the CS VM Cluster machines. You will be using all your VMs for the demo. The VMs do not have persistent storage, so you are required to use git to manage your code. To access git from the VMs, use the same instructions as MP1.

Demo: Demos are usually scheduled on the Monday right after the MP is due. The demos will be on the CS VM Cluster machines. You must use up to the max VMs for your demo (details will be posted on Piazza closer to the demo date). Please make sure your code runs on the CS VM Cluster machines, especially if you've used your own machines/laptops to do most of your coding. Please make sure that any third party code you use is installable on CS VM Cluster. Further demo details and a signup sheet will be made available closer to the date.

Typical tests in the demo will involve checking Fair-Time Inference (make sure you have commands for printing on-screen statistics of query rate ~~query time processing~~ for each job – average, standard deviation, median, 90th percentile, 95th percentile, 99th percentile, as well as number of queries processed so far), and Fault-tolerance (TA will make coordinator fail and ask to see that the inference is proceeding normally; then TA will make up to 3 ~~non-coordinator~~ VMs or coordinator VM fail and ask to see that the inference is proceeding normally).

Language: Choose your favorite language! We recommend C/C++/Java/Go/Rust/Python.

Report: Write a report of no more than 3 pages (12 pt font, typed only - no handwritten reports please!). Briefly describe your design (including architecture and programming framework) for IDunno in less than 1 page (try to draw a figure that captures your main architecture/design decisions). Be concise and clear. Provide the following data and discuss your salient findings: 1) Fair-Time Inference: 1a) When a job is added to the cluster (1 → 2 jobs), what is the ratio of resources that IDunno decides on across jobs to meet fair-time inference?, 1b) When a job is added to the cluster (1 → 2 jobs), how much time does the cluster take to start executing queries of the second job? 2) After failure of **one** (non-coordinator) VM, how long does the cluster take to resume "normal" operation (for inference case)?, 3) After failure of the Coordinator, how long does the cluster take to

resume "normal" operation (for inference case)? Unless otherwise mentioned all experiments are with multiple jobs in the cluster.

Discuss your plots, don't just put them on paper, i.e., discuss trends, and whether they are what you expect or not (why or why not). (Measurement numbers don't lie, but we need to make sense of them!) Stay within page limit. Make sure to plot average, standard deviations, etc.

Submission: There will be a demo of each group's project code. On Gradescope submit your report by the deadline (11.59 PM Sunday). In the git, submit by the deadline your report as well as working code; please include a README explaining how to compile and run your code. Other submission instructions are similar to previous MPs (submit the Google form with your git link, submit the report on Gradescope, and signup for demo, and attend demo).

When should I start? Start now on this MP. Each MP involves a significant amount of planning, design, and implementation/debugging/experimentation work. Learning how to run any open-source system is also a time-consuming task, so please devote enough time to do this. Do not leave all the work for the days before the deadline – there will be no extensions.
Please note that the submission deadline is right at the end of thanksgiving break – please plan appropriately.

Evaluation Break-up: Demo [60%], Report (including design and plots) [30%], Code readability and comments [10%].

Academic Integrity: You cannot look at others' solutions, whether from this year or past years. We will run Moss to check for copying within and outside this class – first offense results in a zero grade on the MP, and second offense results in an F in the course. There are past examples of students penalized in both those ways, so just don't cheat. You can only discuss the MP spec and lecture concepts with the class students and forum, but not solutions, ideas, or code (if we see you posting code on the forum, that's a zero on the MP).

We recommend you stick with the same group from one MP to the next (this helps keep the VM mapping sane on IT's end), except for exceptional circumstances. We expect all group members to contribute about equivalently to the overall effort. If you believe your group members are not, please have "the talk" with them first, give them a second chance. If that doesn't work either, please approach Indy.

Happy Machine Learning (from us and the fictitious MartWall Inc.)!