Language and runtime for parameter servers

- Student: Guobao LI [guobao1993.li@gmail.com]
- Mentor: Matthias Boehm

About me

I'm Guobao LI, third year master at Computer Science in South China University of Technology. I participate in a student exchange project in France. And I just finished my master degree of Computer Science in Polytech Nantes. During this time, I have made two internships in France. The first one was a three-month internship in a start-up as Java backend developer. The second one was a six-month internship in Atos as full stack developer with Java and Angular working on an open source project Alien4Cloud which gives me the inspiration for joining the world of open source. For my part, I like coding and learning the fresh things such as recently blockchain and machine learning. To conclude, I'm a person who cannot stop thinking about tricky problems even when sleeping.

Background

I want to contribute to Apache SystemML project by implementing one of the proposal ideas, namely "[SYSTEMML-2083] Language and runtime for parameter servers". Apache SystemML is a machine learning oriented platform which aims to execute the R-like script, namely dml, in a runtime backend determined by individual physical operators, e.g., a matrix concatenation operator executed in Spark distributed backend and a print operator executed in local backend. It allows to execute machine learning scripts in large-scale environment without rewriting scripts to adapt to the specific computation framework such as Spark. Apache SystemML is already well equipped to execute the parallel computation by distributing workloads to the workers who work on its own data partition in a same or different logical manner. The idea implies to complement the two existing execution strategies (data-parallelism and task-parallelism) by language and runtime primitives for parameter servers via creating a third new execution strategy, namely "model-parallel computation" which distributes a large model to workers and trains the model in a parallel manner. Concretely, a new built-in function "paramserv" will be created by extending the language support and implementing the runtime for stateful parameter server and its common push/pull primitives inside SystemML. Since SystemML already provides the large-scale data manipulation over Spark, we can simply concentrate on implementing the parameter update strategies (synchronous, asynchronous, hogwild!, stale-synchronous). To contribute for this big project, I'd like to focus on implementing the extension of language support, the primitive of data-parallel parameter server containing its push/pull methods and the synchronous update strategy on local multi-threaded and Spark distributed runtime backend but leave the checkpointing strategies and other update strategies as option, i.e., asynchronous, hogwild!, stale-synchronous. The further implementation for the primitive of model-parallel parameter server and an automatic selection of execution backend will also be optional.

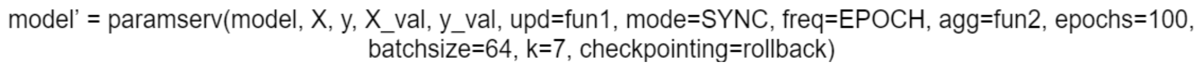
Design / description of work

In the context of SystemML, the "paramserv" built-in function, a stateful function, will be able to update the model parameters by calling the dml stateless function, e.g., the parameters update function and aggregate function, with a predefined frequency. That makes it a second-order function which can orchestrate the user-defined function call inside the operator. The "paramserv" function will be able to support the data-parallel execution. When needing to train the model with a large training dataset, it will split this large dataset into small data partitions according to the "degree of parallelism", i.e., the number of workers, and distribute this splitted dataset across the workers so that they can call the user-defined statistical function to work on its local portion of dataset in a parallel manner. And then the result of each worker should be aggregated to update the global parameters. The global parameters are often saved in parameter server. Somehow, they would be too large to fit in one single parameter server. Hence, if time permits, "model-parallelism" will be put forth to distribute this large model parameter into some small disjoint or overlapping parameters so that the workers can work on its assigned portion of parameters in a parallel manner. For example, in the context of training neural network, the updating workloads of a layer or sub-layers can be distributed to different workers. And then the workers can exchange the weights of layer via a centralized parameter server.
The "paramserv" function will support two major execution backend for runtime, i.e., local multi-threaded and Spark distributed. However, the user need to determine which execution backend should be applied. And if time permits, we will further support automatic selection for the execution backend. With the help of the configuration, the "paramserv" function can use the "update frequency" to determine how often the parameters will be updated. If the update frequency is "per batch", the parameters are updated after every function call of calculating intermediates such as gradients, whereas if its per epoch, we call the function once per batch with the parameters returned by the previous call and only update the global parameters once we made a full pass over the data. It can use the "update strategies" to determine in which way the global parameters will be updated. For example, if the parameters update is done in "synchronous" way, after a batch or an epoch each worker could not continue to the next batch or epoch until it receives a copy of global updated parameters, i.e., the local parameters of each worker should be synchronized with the server in some period. The "paramserv" function could use the "degree of parallelism" to determine the number of workers. It can also use the "checkpointing" to determine how and when the state of "paramserv" function should be persisted in some high available database. We

should consider that the “paramserv” function could run for a long period with limited downtime and be able to recovery from panic. Hence, it will be necessary to save the metadata of the “paramserv” function, e.g., the configuration and runtime snapshot so that we could recover the function to the precedent state. For the real data, i.e., the parameters, SystemML has direct access to the underlying distributed file system (e.g., HDFS or object storage), and we would simply periodically write it out. Hence, we do not need the checkpointing for real data. And then the following parts will show the details about the constitution of work.

● API design of the paramserv built-in function (essential)

The objective of “paramserv” built-in function is to update an initial or existing model with configuration. An initial function signature is illustrated in Figure 1. We are interested in providing the model, the training features and labels, the validation features and labels, the batch update function, the update strategy (e.g. sync, async, hogwild!, stale-synchronous), the update frequency (e.g. epoch or batch), the aggregation function, the number of epoch, the batch size, the degree of parallelism as well as the checkpointing strategy (e.g. rollback recovery).



```
model’ = paramserv(model, X, y, X_val, y_val, upd=fun1, mode=SYNC, freq=EPOCH, agg=fun2, epochs=100,
                    batchsize=64, k=7, checkpointing=rollback)
```

Figure 1: Signature of “paramserv” function

● Implement the compiler extension of the language support for the paramserv built-in function (essential)

This part aims to add an additional language support for the “paramserv” function in order to be able to compile this new function. Since SystemML already supports the parameterized builtin function, we can easily extend an additional operation type and generate a new instruction for the “paramserv” function. Recently, we have also added a new “eval” built-in function which is capable to pass a function pointer as argument so that it can be called in runtime. Similar to it, we would need to extend the inter-procedural analysis to avoid removing unused constructed functions in the presence of second-order “paramserv” function. Because the referenced functions, i.e., the aggregate function and update function, should be present in runtime.

● Design of parameter server primitive

● Data-parallel parameter server (essential)

Parameter server allows to persist the model parameters in a distributed manner. It is specially applied in the context of large-scale machine learning to train the model. The parameters computation will be done with data parallelism across the workers. The data-parallel parameter server architecture is illustrated in Figure 2. With the help of a lightweight parameter server interface [1], we are inspired to provide the push and pull methods as internal primitives, i.e., not exposed to the script level, allowing to exchange the intermediates among workers. In the architecture, there are two types of nodes: the parameter server, which will be a single node or a cluster of nodes to save the global parameters, and workers, which load a partition of data, calculate and push the intermediates and then pull the global parameters [2].

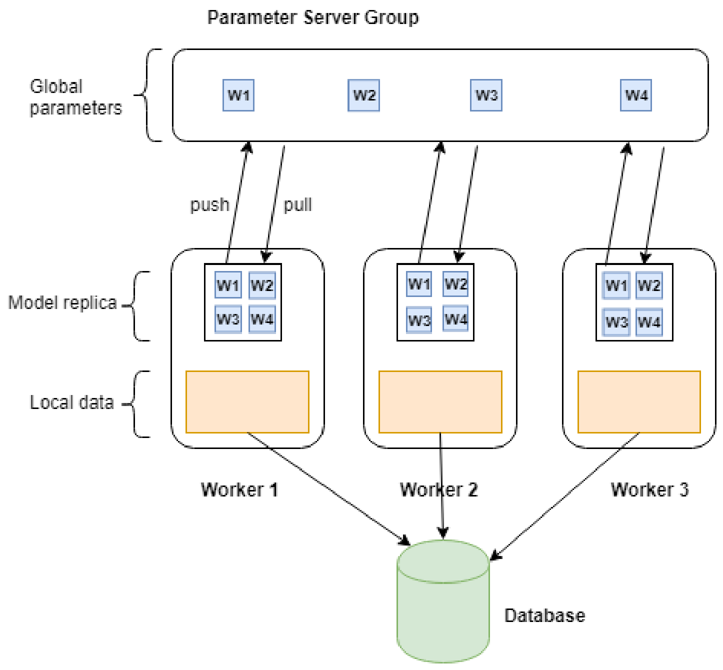


Figure 2: Architecture of data-parallel parameter server

- Model-parallel parameter server (optional)

  When the model as a set of weight matrices is too big to fit a memory of single node, it will be splitted into portions of matrices and be saved across different parameter servers forming a group of parameter server. A portion or several portions of parameters will be assigned to each worker which operates and caches parameter updates locally. This model-parallel parameter server architecture is illustrated in Figure 3.

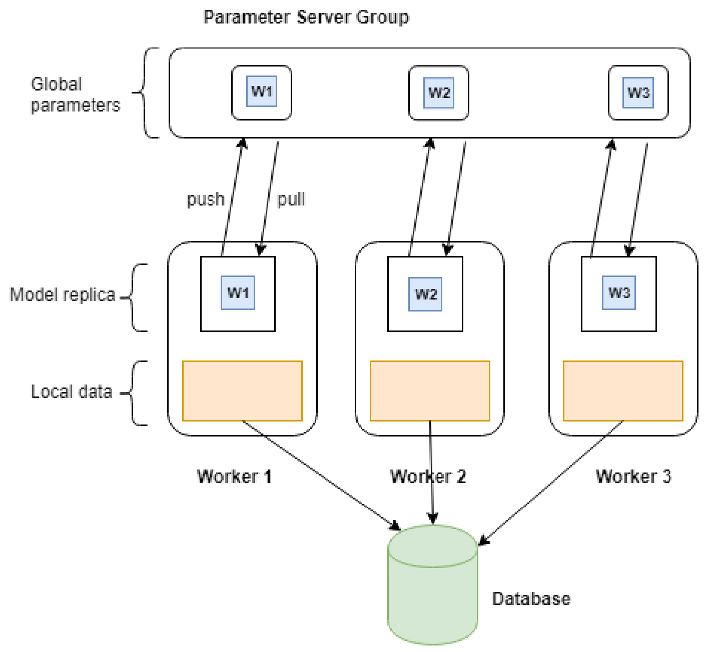


Figure 3: Architecture of model-parallel parameter server

For the model-parallel parameter server, the global parameters are splitted and copied across the workers. Each worker can individually compute intermediates, push them to parameter server which persists this model replica. And then this server will aggregate intermediates and update the local portion of parameters. And finally it broadcasts the new updated parameters to the worker.

- Single-node parameter server (essential)

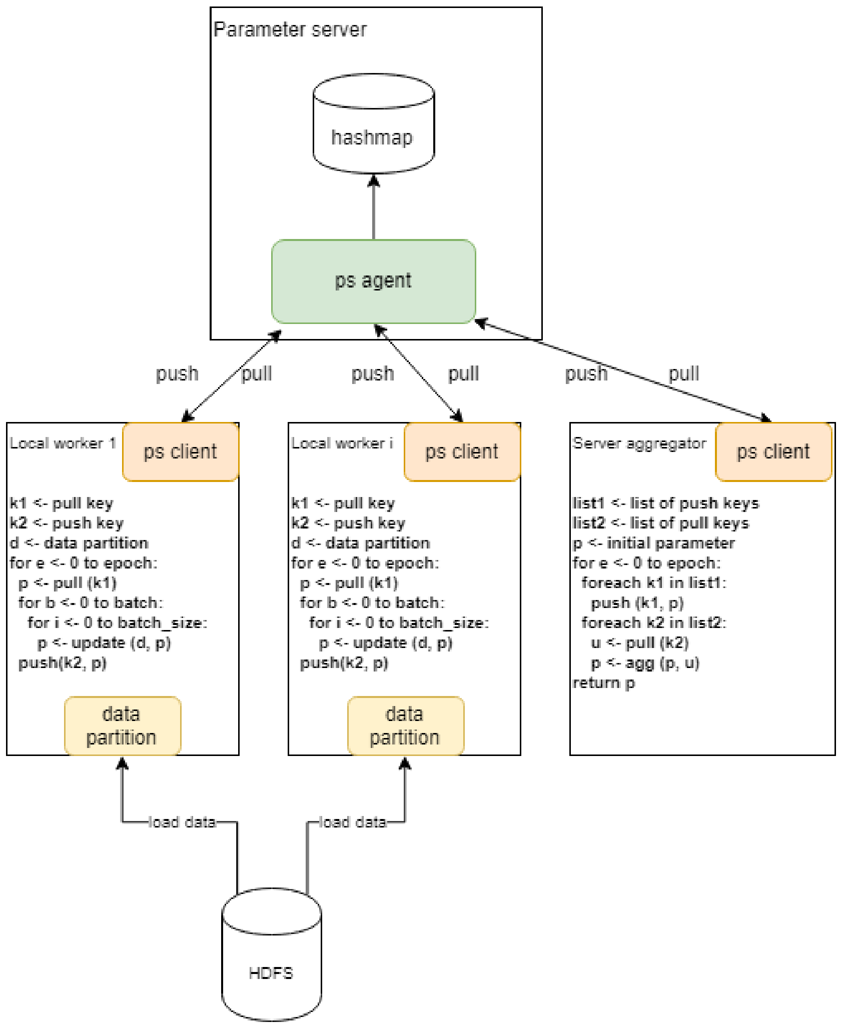
  I have two ideas to implement the parameter servers. The first one is to create a single-node parameter server by maintaining a hashmap inside the CP (Control Program) where the parameter as value accompanied with a defined key. For example, inserting the global parameter with a key named “worker-param-replica” allows the workers to retrieve the parameter replica. Hence, in the context of local multi-threaded backend, workers can communicate directly with this hashmap in the same process. And in the context of Spark distributed backend, the CP firstly needs to fork a thread to start a parameter server which maintains a hashmap. And secondly the workers can send intermediates and retrieve parameters by connecting to parameter server via TCP socket. Since SystemML has good cache management, we only need to maintain the matrix reference pointing to a file location instead of real data instance in the hashmap.

- Multi-node parameter server (optional)

  The second one is to create a multi-node parameter server which allows to distribute a large model (that would contain billions of individual weights) in a coarse grained manner of matrices. Hence, we could simply broadcast a dictionary (of which matrix is on which parameter server) to all workers because this dictionary will be very small and read only to retrieve metadata of the appropriate parameter server. Based on the dictionary a worker would simply transfer the specific weight matrices to the appropriate parameter server.

Design and implement the local multi-threaded execution backend (essential)

This part aims to design and implement a local execution backend for the compiled “paramserv” function. It consists of the implementations of partitioning the data for worker threads, launching the single-node parameter server, shipping and calling the compiled statistical function and creating different update strategies. We will focus on implementing BSP execution strategies [2], i.e., synchronous update strategy including per epoch and per batch. And other update strategies (e.g. asynchronous, stale-synchronous) and checkpointing strategies should be optional and will be added if time permits. The architecture for synchronous per epoch update strategy is illustrated below in Figure 4.



```
k1 <- pull key
k2 <- push key
d <- data partition
for e <- 0 to epoch:
  p <- pull (k1)
  for b <- 0 to batch:
    for i <- 0 to batch_size:
      p <- update (d, p)
  push(k2, p)
```

```
k1 <- pull key
k2 <- push key
d <- data partition
for e <- 0 to epoch:
  p <- pull (k1)
  for b <- 0 to batch:
    for i <- 0 to batch_size:
      p <- update (d, p)
  push(k2, p)
```

```
list1 <- list of push keys
list2 <- list of pull keys
p <- initial parameter
for e <- 0 to epoch:
  foreach k1 in list1:
    push (k1, p)
  foreach k2 in list2:
    u <- pull (k2)
    p <- agg (p, u)
return p
```

Figure 4: Architecture of PS with synchronous per epoch update strategy

My idea is to create a hashmap in CP and fork a thread to launch single-node local parameter server which provides the push/pull service. Each long-run worker launches a local PS client to send and retrieve parameters. After that, it loads data partition, operates the parameter updating per batch, pushes its local updated parameter and retrieves a new global parameter. And then a server aggregator thread will be spawned to retrieve the local updated parameters of each worker using the related keys in a round robin way, aggregate the parameters and push the new global parameter with the parameter related keys. At last, the paramserv function main thread should wait for the server aggregator thread joining it and got the last global parameters as final result. Hence, the pull/push primitive methods can bring more flexibility and facilitate to implement other update strategies.

- ## Design and create the Spark distributed execution backend (essential)

This part aims to design and implement a Spark distributed execution backend for the compiled “paramserv” function. It consists of the implementations of partitioning the data for worker node, the single-node parameter server, shipping and calling the compiled statistical function and different update strategies. With the handles of JavaPairRDD, we can easily split the training data according to the number of workers. And then we can ship the compiled function by implementing the instance of transformation function of RDD. Inside the transformation, the parameter update task can be executed. Workers communicate with the parameter server via TCP. Similar to the local multi-threaded backend, the final result can be obtained when the server aggregator finishes.

- Integration test (essential)

  This part aims to implement the integration test for the “paramserv” function.

- Implementation of sample (essential)

  This part aims to implement a sample dml script applying the “paramserv” function.

- Documentation (essential)

  This part aims to add the documentation for the “paramserv” function.

Deliverables

Below I tried to structure the contributions to SystemML project as part of GSoC program into several core steps:

- API design for the new paramserv built-in function
- Compiler extension for the new paramserv built-in function
- Design of the single-node data-parallel parameter server primitive
- Implementation of local execution backend
- Implementation of Spark execution backend
- Integration test for the new paramserv built-in function
- Implementation of sample dml script using the new paramserv built-in function
- Documentation for the new paramserv built-in function

Scheduling

Following is an approximate schedule for accomplishing important milestones. I provided an estimated end date for each task along with their expected duration. There is a long period of time until the coding period starts (proposal selection period + community bonding period), which I plan to use to get familiar with particular the module relevant to the execution runtime both local and distributed mode, continuing to contribute to SystemML project (such as following the issue [SYSTEMML-2077] for which I made the PR). I expect after this period to have a very clear understanding of how the project should be developed.

- May 14: End of bonding period, start of GSoC program

- May 21: API design of the paramserv built-in function (1 week)
- May 28: Implementation of compiler extension for paramserv built-in function (1 week)
- June 4: Design of single-node data-parallel parameter server primitive(1 week)
- June 15: First evaluation (5 days)
- July 25: Initial version for the implementation of local execution backend (2 week)
- July 9: Initial version for the implementation of Spark execution backend (2 week)
- July 13: Second evaluation (5 days)
- August 6: Second version for the backend implementations (4 week)
- August 13: Integration test, implementation of samples and documentation (1 week)
- August 14: Submit final product

Other commitments

I have no other commitments. For the GSoC program, I plan to allocate at least 30 hours per week, in a flexible way.

Community engagement

In order to get a global view of the project SystemML, I have at first gone through the documentation of project and launched the samples as a common user. Then I have followed the given instructions in JIRA, read the white paper of SystemML to get further information and made some exchanges with Matthias to get responses to my questions. Furthermore, to get a better understand and more technical details, I have made a PR for the issue [SYSTEMML-2077] and got accepted. (That makes me so motivated.) And on github for this PR, I got much inspiration through several public exchanges with Matthias. Concretely, I have added a new “eval” built-in function which allows to take a function pointer as argument and to call the function with the rest of the arguments as its parameters. It is similar to the “paramserv” built-in function which should be a second-order function that takes the actual training function per batch as an argument. Hence, it facilitates to extend the compiler for the new “paramserv” built-in function. Meanwhile, working on this issue helps me to know about the dml compiler and the single-node runtime.

References

[1] https://github.com/dmlc/ps-lite
[2] Jiang J, Cui B, Zhang C et al. Heterogeneity-aware distributed parameter servers . In: Proceedings of SIGMOD Conference . 2017