MLMS – Architecture Specification

Author(s)

Sriram Bhamidipati

Table of Contents

[Overview 2](#_Toc71923164)

[Reference(s) 2](#_Toc71923165)

[MLMS Block Diagram 2](#_Toc71923166)

[Detailed explanation 3](#_Toc71923167)

[MLMS Controller 3](#_Toc71923168)

[ML Application process (MLAP) 3](#_Toc71923169)

[Logging/Debugging 4](#_Toc71923170)

[REST Interface(s) 4](#_Toc71923171)

[Salient notes 5](#_Toc71923172)

[Potential Improvements 5](#_Toc71923173)

# Overview

The MLMS is implemented as a 2-tier application where in the controller is the application entry point and acts as an initializer and which spawns the configured ML Application Processes (henceforth referred as MLAP) There will be as many MLMPs as the ML Application instances (one-one correspondence)

# Reference(s)

* MLMS-SRS.docx
* MLMS-HighLevelRequirements.md

Assumptions

1. All ML applications are homegrown (even though the core algorithms may be imported from elsewhere (eg scikitlearn etc)

### MLMS Block Diagram

Rest API Server

Figure 1 MLMS System Block Diagram

Rest client(s)

Mon DB

# Detailed explanation

Please refer to Figure 1

## MLMS Controller

MLMS Controller is the entry point and it Initializes with run time parameters such as

1. DB connection details (Type of DB Mysql, postgress SQLServer, user authentication credentials as applicable)
2. Loads the DB and fetches the details from tables such as ML application connection details
3. Debug Logging details
4. Reads initialization file for other configurable parameters
5. Spawns the ML monitoring app as per the class it belongs to as a thread
6. Initializes additional services such as web server (for RESTful interface)

All MLAP processes connect to the DB:

1. Any REST API (if available)
2. DB (if available)

MLSMController() # Pseudocode

{

Initialize

DB connection

Start Restful Server

Spawn ML App (MLAP) processes

}

## ML Application process (MLAP)

Key Steps for a monitoring application

CheckRules()

{

Run checks for various specifics\*:

* Detect and report Dependency changes
* Data invariants hold in training and serving inputs, i.e. monitor Training/Serving Skew
* Training and serving features compute the same values
* Models are not too stale
* The model is numerically stable
* The model has not experienced dramatic or slow-leak regressions in training speed, serving latency, throughput, or RAM usage
* The model has not experienced a regression in prediction quality on served data
* <Add more>

Update DB (eg Redis )

}

\*each rule checker may also perform an optional corrective action as per the business logic

Model.run()

{

Forever {

CheckRules()

ServiceExternalCommands()

}

Sleep(<configured periodicity can be specific to model or a global default, say 1 minute>) seconds

Each ML application process is run as an independent process and autoscaling can be easily achieved by some rule based actions

# Logging/Debugging

Easy to debug/troubleshoot. Dump routines and other instrumentation will be provided

Configurable log level (default : notice)

MLMP will log to a separate file

All check functions are extensively logged for debugging

Dynamic debug flag attributes are available to change debug levels

Audit trail is present for accountability using RBAC

# REST Interface(s)

# Highlights

1. The approach is very loosely coupled, distributable and hence highly scalable
2. Highly extensible for future ML performance requirements (can easily add/omit checker functions with minimal changes and minimal deployment time)
3. Auto scalable, for optimal resource (cpus/memory) usage, saving memory
4. Loosely coupled with interchangeable subsystem(s) for in memory databases, DBMS
5. Persistent (and hence fault tolerant)
6. A given monitoring app (MLMP) can quicky respawn, making the overall system Highly available
7. Provision for extensive debugging and troubleshooting
8. Rate limiting for API server, preventing any performance degradation/vulnerabilities/attacks (leaky bucket)

# Improvements/TODOs

* Implementing Configuration DB. Such an approach is more comprehensive/robust
* Not implemented in a distributed manner but can be easily extended to be so.

There can be several variants possible for a distributed MLMS system

* + Distributed MLAPs (supporting a slew of ml algorithms running across several computes
  + Distributed MLMS (agent/master mode). This will be an extreme case of overcrowded scenario
* Best practices of Pythonic coding/building to make it production ready and maintainability
* Performance issues in highly scaled scenarios