memPrediction 用户文档

Product Name : memPrediction

Product Version : V1.1

Release Date : 2026.02

Contact : @张静文 （zhangjingwen.silvia@picoheart.com）

@李艳青 （liyanqing.1987@picoheart.com）

一、简介

LSF 是 IBM 旗下的一款分布式集群管理系统软件，负责计算资源的管理和批处理作业的调度。它具有良好的可伸缩性和高可用性，支持几乎所有的主流操作系统，通常是高性能计算环境中重要的基础软件。

LSF 使用过程中一个常见的痛点是，用户不清楚每个任务的具体资源需求，尤其是 memory 资源需求（过量使用容易造成操作系统 OOM），不设/多设/少设资源预占量，都会造成服务器之间的负载不均衡，进而影响计算任务的运行效率。

memPrediction 就是用来解决这一问题的工具，它基于机器学习的方法对用户提交的 job 的 max memory 进行预测，通过 LSF esub 的方式自动对未设置 memory reservation 的 job 增加 resource rusage 的设置，从而达到 LSF 资源（memory）自动预设的目的。

memPrediction 的主要功能如下。

- **数据采集(sample)**

采集 lsf 中的 job 信息， 用于数据分析和模型训练。

- **数据分析(report)**

对 job 的 memory/cpu 预留情况进行分析， 并提供分析报告。

- **模型训练(train)**

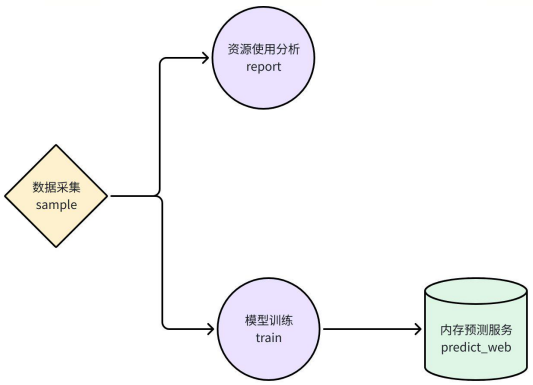
使用采集到的 job 信息， 使用机器学习的方法，训练用于 job max memory 预测的模型。

- **memory 预测(predict)**

使用训练好的模型， 在 job 提交时候对其 max memory 进行预测， 可以使用 **调用脚本/API** 的方式获取结果。

- **web 数据展示**

提供 web 应用， 可以查看 LSF 中 JOB 总体的 memory 预留情况， 以及单个用户的 JOB 信息和 JOB Memory 信息。



二、环境依赖

2.1 操作系统依赖

memPrediction 的开发和测试操作系统为 **CentOS Linux release 7.9.2009 (Core)**，这也是 IC 设计常用的操作系统版本之一。

centos7/centos8/rocky8/redhat7/redhat8 理论上都可以运行，主要的潜在风险在于系统库版本差异可能会影响部分组件的正常运行。

2.2 python 版本依赖

memPrediction 基于 python 开发，其开发和测试的 python 版本为 **python3.12.12**。

不同版本的 python 可能会有 python 库版本问题，按照系统要求安装对应版本的 python 库即可解决。

2.3 集群管理工具

memPrediction 依赖 LSF 集群管理系统，暂不支持其它集群管理系统。

2.4 硬件要求

该工具进行模型训练时， 对机器的内存有一定要求。 根据最大训练数据量的不同（最大训练数据量可以在安装配置时候修改， 默认为 10,000,000 条）， 内存的要求也不同，**务必不要使用虚拟机进行训练**。以下是根据以往训练经验的推荐值：

- 10,000,000 条（默认值）：推荐训练机器内存 2TB，至少 1.5TB。
- 5,000,000 条：推荐训练机器内存 1.5TB，至少 1TB。
- 2,000,000 条：推荐训练机器内存 1TB，至少 768GB。

模型部署对机器也有一定要求，主要是 cpu 方面的要求，推荐核数 n >= 4，否则同时预测的 job 过多的话，可能导致机器负载过重，无法在规定时间内返回预测结果。

三、工具安装与部署

3.1 工具安装

工具安装之前，首先参照第二章“环境依赖”满足的 memPrediction 环境依赖关系。

安装包下的文件和目录如下。

```
Bash
[root@ic-admin2 memPrediction]# ls -p
bin/  common/  config/  db/  install.py  lib/  tools/  web_app/
```

确认 python 版本正确（Python 3.12.12）。

```
Bash
[root@ic-admin2 memPrediction]# python3 --version
Python 3.12.12
```

install.py 的帮助信息如下。

```
Bash
[root@ic-admin2 memPrediction]# python3 install.py -h
usage: install.py [-h] [-p PREFIX] [-c] [-f]

options:
  -h, --help            show this help message and exit
  -p PREFIX, --prefix PREFIX
                        Specify lsfMonitor install path on config
file, default is current directory.
  -c, --clean           Cleanup old installation.
  -f, --force           Install by force.
```

- **prefix（-p）**：指定 memPrediction 的安装路径， 默认安装在当前路径下。
- **clean（-c）**：指定该参数时， 清理所有安装时生成的脚本和路径， 将路径恢复到没有安装的情况。
- **force（-f）**：在同一个路径下重复进行安装时， 可以指定 -f 参数， 将重新生成 config.py 等配置文件。默认为 False。

在安装目录下，使用命令 `python3 install.py` 安装 memPrediction。（公共软件安装一般需要使用 root 账号，当然，仅本人使用用私人账号安装亦可）

```
Bash
[root@ic-admin2 memPrediction]# python3 install.py
Generate new
/ic/data/usr/liyanqing.1987/tools/memPrediction/tools/predict_gcon
f.py ...
Generate new
/ic/data/usr/liyanqing.1987/tools/memPrediction/tools/esub.mem_pre
dict ...
>>> Check python version.
    Required python version : (3, 12)
    Current  python version : (3, 12)

>>> Generate script
"/ic/data/usr/liyanqing.1987/tools/memPrediction/bin/sample".
>>> Generate script
"/ic/data/usr/liyanqing.1987/tools/memPrediction/bin/report".
>>> Generate script
"/ic/data/usr/liyanqing.1987/tools/memPrediction/bin/train".
>>> Generate script
"/ic/data/usr/liyanqing.1987/tools/memPrediction/bin/predict".
>>> Generate script
"/ic/data/usr/liyanqing.1987/tools/memPrediction/tools/update".
>>> Generate script
"/ic/data/usr/liyanqing.1987/tools/memPrediction/web_app/setup".
>>> Generate script
"/ic/data/usr/liyanqing.1987/tools/memPrediction/web_app/backend/d
ataCollector".
>>> Generate script
"/ic/data/usr/liyanqing.1987/tools/memPrediction/tools/.env".
>>> Generate config file
"/ic/data/usr/liyanqing.1987/tools/memPrediction/config/config.py"
.
>>> Generate script
"/ic/data/usr/liyanqing.1987/tools/memPrediction/tools/predict_web
.service".
>>> Generate script
"/ic/data/usr/liyanqing.1987/tools/memPrediction/tools/stopservice
.sh".
>>> Generate script
"/ic/data/usr/liyanqing.1987/tools/memPrediction/tools/train.sh".
>>> Generate script
```

```
"/ic/data/usr/liyanqing.1987/tools/memPrediction/tools/web_startup
.sh".

Done, Please enjoy it.
```

如果采样用户跟安装用户不一致，记得将 db 路径权限开放为 777，否则会导致写入错误的问题。

```
Bash
[root@ic-admin2 memPrediction]# chmod 777 db -R
```

3.2 工具配置

config/config.py：安装目录下主要的配置文件为 config/config.py，用于配置工具的一些基本设置。其中除了用于预测的模型路径， 都已经在安装的时候写入了默认配置， 可以根据需要进行修改。

```
Python
# job infomation database save directory, format: csv/sqlite.
db_path =
"/ic/data/usr/liyanqing.1987/tools/memPrediction/db/job_db"

# Specify job database format
job_format = 'csv'

# job rusage analysis report template
report_template =
"/ic/data/usr/liyanqing.1987/tools/memPrediction/config/rusage_rep
ort_template.md"

# job rusage analysis report db path
report_path =
"/ic/data/usr/liyanqing.1987/tools/memPrediction/db/report_db"

# training job memory model config yaml file
training_config_yaml =
"/ic/data/usr/liyanqing.1987/tools/memPrediction/config/training.c
onfig.yaml"
```

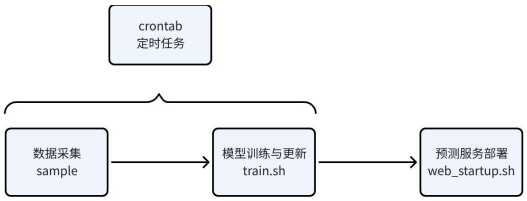
```
# train and save model this directory
model_db_path =
"/ic/data/usr/liyanqing.1987/tools/memPrediction/db/model_db"

# prediction model config yaml
predict_model =
"/ic/data/usr/liyanqing.1987/tools/memPrediction/db/model_db/lates
t"

# model training max lines, default 10,000,000. if set to '0' or
'', means infinity.
max_training_lines = 10000000
```

- **db_path**: 数据库路径， 用于存放收集到的 lsf job 的信息， 也是进行 job 数据分析和 memory 模型训练的默认数据库路径。**如果不使用 memPrediction 的采样工具， 而是使用 lsfMonitor 的采样工具采集到的信息， 这里需要修改为 lsfMonitor 采集 job 信息后存放 job 信息的路径。**
- **job_format**: 采集到的 job 信息的格式， 默认为 'csv'。如果使用 memPrediction 提供的 job 采集的工具进行采集， 这一项不需要修改； **如果想要使用 lsfMonitor 的采样工具采集到的信息， 这里需要修改为 'json'。**
- **report_template**: 数据分析报告模板（markdown 格式）， 可以使用该模板生成一份默认的数据分析报告， 也可以使用提供的变量生成自己想要的数据分析报告，在后文中会介绍提供的变量。
- **report_path**: 存放数据分析报告的路径。
- **training_config_yaml**: 训练模型的配置文件， 已经内置了一份默认的模型训练配置信息， 也可以自行进行微调。
- **model_db_path**: 存放训练好的模型文件的路径。
- **predict_model**： 用于 memory 预测的模型存放位置， 调用预测脚本的时候会使用到。需要先训练一个模型，再把该模型的路径写到配置文件中， 作为预测脚本的默认模型。默认情况下会使用 model_db_path 下的 latest model。
- **max_training_lines**： 训练的最大数据量（条数）， 可以修改， 推荐至少在 2,000,000 条以上，默认为 10000000。请保证训练机器的 memory 充足，保证训练过程可以完成， 具体**可参考 第二章 环境依赖中的 2.4 硬件要求**。如果这里不填或者置为 0 的话， 默认为训练全部数据， 为了防止训练过程中 OOM 的情况， 强烈建议不要置为 0 。

3.3 快速上手指南

工具安装完成后， 可以参考以下步骤进行快速部署使用。如果需要了解工具的使用细节和优化方法， 可以在第四章节中查看。以下内容中，其中标注了 **required** 是安装部署该预测必须的项， **optional** 是可选项。

如下图所示， 必须执行的步骤有三步， 数据采集和模型训练需要部署为定时任务， 推荐在定时任务开启一个月后， 就可以进行预测服务的部署了。



3.3.1 定时采样部署（required）

该工具的训练数据依赖于定时采集的 job 信息， 可以通过 crontab 的方式配置定时采集任务， 收集 job 数据， 进行训练和分析。sample 工具更详细的用法请参考 **第四章第一小节 sample: 数据采集工具**， 这里仅记录快速上手配置方法。

crontab 中默认是没有任何环境的，所以需要在 crontab 中设置好 PATH 等变量，否则 sample 中引用的 bjobs 等工具无法生效。crontab 需要配置的内容如下：

- **SHELL**: 通过 `echo $SHELL` 获得， 一般为 `SHELL=/bin/bash`
- **PATH**：通过 `echo $PATH` 获得，需要保证环境变量中已经有 lsf（例如， 如果需要 module load lsf 才能使用 bsub， 需要先 module load lsf）。
- **LSF_***：通过 `env | grep LSF` 获得，如果不设置会导致 crontab 中 LSF 相关的命令无法执行。
- **定时任务**： `59 22,23 * * *  ${INSTALL_PATH}/bin/sample -c`， 需要将其中的 ${INSTALL_PATH} 更改为安装路径， 定时任务实际上在每天 23：59 执行一次即可，为防止意外我们多采样一次。

以下是一个 crontab 的范例：

```Bash
[root@ic-admin2 memPrediction]# crontab -l
SHELL=/bin/bash
PATH=/ic/software/tools/python3/3.12.12/bin:/ic/software/tools/lsf
/10.1/linux3.10-glibc2.17-
```

```
x86_64/etc:/ic/software/tools/lsf/10.1/linux3.10-glibc2.17-
x86_64/bin:/bin:/sbin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/us
r/bin
LSF_SERVERDIR=/ic/software/tools/lsf/10.1/linux3.10-glibc2.17-
x86_64/etc
LSF_LIBDIR=/ic/software/tools/lsf/10.1/linux3.10-glibc2.17-
x86_64/lib
LSF_BINDIR=/ic/software/tools/lsf/10.1/linux3.10-glibc2.17-
x86_64/bin
LSF_ENVDIR=/ic/software/tools/lsf/conf

# For memPrediction, sample job info.
59 22,23 * * * /ic/software/tools/memPrediction/bin/sample -c
```

注：

memPrediction 兼容 lsfMonitor 中的 bsample 工具进行的 LSF job 信息采样，如果已经使用了 bsample -j 进行 job 信息的采样， 那么可以直接在配置文件 config.py 中修改如下两个值， 从而直接使用 lsfMonitor 采样的信息， 就不需要使用 memPrediction 自带的采样工具了。

- **db_path**：修改为 bsample 采样的 job 信息的路径
- **job_format**：默认为 csv，如果要使用 bsample 工具采样的信息， 这里可能需要修改为 'json' / 'sqlite'
  - 如果 lsfMonitor 采样到的数据是 json 格式， 那么需要将可以将 job_format 修改为 'json'

```sql
...
# job infomation database save directory, format:
csv/sqlite.
db_path =
"/ic/software/cad_data/it/lsfMonitor/db/IC_CLUSTER/job"

# Specify job database format
job_format = 'json'
...
```

  - 如果 lsfMonitor 采样到的数据保存到的 sqlite 数据库中， 那么需要将 job_format 修改为 'sqlite'

```sql
...
# job infomation database save directory, format:
csv/sqlite.
db_path =
"/ic/software/cad_data/it/lsfMonitor/db/IC_CLUSTER/job"

# Specify job database format
job_format = 'sqlite'
...
```

3.3.2 定时分析部署（optional）

采集到的 job 信息， 可以用于 job 的 memory/cpu 资源使用方面的分析。memory 分析会生成一份 markdown 报告， 两种分析都会生成若干分析图表。这一步对于模型预测来说， 不是必须的， 而仅仅提供资源使用率方面的参考。

如果需要定时进行 memory 和 cpu 方面的分析报告， 可以将 report 也配置为定时任务，以便生成定期的 job 资源使用分析报告。report 工具更详细的用法请参考 **第四章第二小节 train: 数据训练工具**， 其中提供了报告模板配置方法和模版的基本样式，这里仅记录快速上手配置方法。

该定时任务的 crontab 不需要额外配置环境变量，仅配置定时任务即可： `0 0 * * 1 ${INSTALL_PATH}/bin/report -m`，需要将其中的 ${INSTALL_PATH} 更改为安装路径。

以下是一个 crontab 的范例， 每周一进行一次分析:

```Bash
# For memPrediction, generate report for memory and cpu analysis.
0 0 * * 1 /ic/software/tools/memPrediction/bin/report -m
0 0 * * 1 /ic/software/tools/memPrediction/bin/report -c
```

3.3.3 定时训练部署（required）

要使用模型的预测功能， 前提是必须要先训练好一个模型。而模型训练的前提是， 数据库路径必须有采集到的数据： 也就是开始定时训练之前， 请确保定时采集任务已经开始。

工具提供 **模型训练脚本 train.sh**， 在安装路径的 tools 目录下可以找到， 可以通过直接执行该脚本的方式，进行模型训练和模型更新。模型训练也可以使用 crontab ， 定期使用采集到的数据进行训练， 推荐的定时任务频率是 **间隔 1 周以上**。

train 工具更详细的用法请参考 **第四章第三小节 train: 数据训练工具**， 其中提供了模型调优的方法和思路，这里仅记录快速上手配置方法。

该定时任务的 crontab 不需要额外配置环境变量，仅配置定时任务即可： `0 0 1 * * ${INSTALL_PATH}/tools/train.sh`，需要将其中的 ${INSTALL_PATH} 更改为安装路径。

以下是一个 crontab 的范例， 每个月 1 号进行一次训练：

```
Bash
# For memPrediction, training.
0 0 1 * * /ic/software/tools/memPrediction/tools/train.sh
```

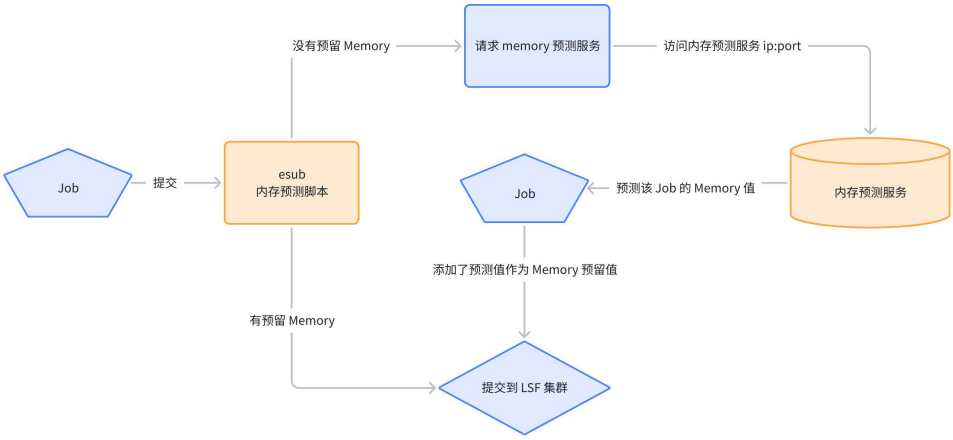
3.3.5 在 LSF 中使用内存预测服务

memPrediction 可在 LSF 作业提交阶段通过 esub 自动调用预测服务。当 Job 未指定 memory 预留时，esub 会请求模型预测推荐值，并将该值写入 Job 的 `rusage[mem]` 作为内存预留值。

该功能由两部分组成：

（1）内存预测服务：提供模型预测 HTTP 服务， 服务需要在一台 LSF 集群可以访问到的机器上启动。

（2）esub 内存预测脚本：在提交时调用内存服务并修改 Job 的 memory 预留参数。



注意： **memPrediction 的 安装和服务部署需要在同一台可以被 LSF 集群访问的机器上进行**。在使用 memPrediction 的模型预测服务期间， 该机器可以重启， 但是不可以销毁机器或更改机器 ip， 否则将会造成模型预测服务失效。

如果需要修改 memPrediction 内存预测服务所在的机器和端口， 可以参考 **4.5 内存预测服务** 修改相关配置信息并重启服务生效。**如果当前有一个以上网络互相隔离的环境， 需要在每个环境中的重新安装部署 memPrediction 服务和配置 esub 内存预测脚本。**

3.3.5.1 部署模型预测 web 服务（required）

由于模型的精准度，和训练数据的分布和量是息息相关的。因此，建议**在数据采集一个月以上**之后，再部署 memory 预测的 web 服务。

工具提供模型预测 **web 服务部署脚本 web_startup.sh**，可以在安装路径在下的 tools 中找到。可以使用该脚本在安装工具的机器上，自动部署一个名为 predict_web 的模型预测 web 服务。

该脚本需要使用 `systemctl` 命令，**请确保执行该脚本的账号具有 root 权限**。

```
Bash
[root@ic-admin2 memPrediction]# tools/web_startup.sh
Start predict web service ...
systemctl enable predict_web
Created symlink from /etc/systemd/system/multi-
user.target.wants/predict_web.service to
/usr/lib/systemd/system/predict_web.service.
systemctl start predict_web
check predict web status...
● predict_web.service - LSF memory prediction web service
   Loaded: loaded (/usr/lib/systemd/system/predict_web.service;
enabled; vendor preset: disabled)
   Active: active (running) since Fri 2024-06-14 11:43:44 CST;
120ms ago
 Main PID: 9445 (gunicorn)
    Tasks: 1
   Memory: 3.5M
   CGroup: /system.slice/predict_web.service
           └─9445 /ic/software/tools/python3/3.12.12/bin/python3.8
/ic/software/tools/python3/3.12.12/bin/gunicorn -c
predict_gconf.py predict_web:app

Jun 14 11:43:44 ic-admin2 systemd[1]: Started LSF memory
prediction web service.
```

部署成功后，可以通过以下命令查看当前服务的状态：

```
Bash
[root@ic-admin2 memPrediction]# systemctl status
```

```
predict_web.service
● predict_web.service - LSF memory prediction web service
   Loaded: loaded (/usr/lib/systemd/system/predict_web.service;
enabled; vendor preset: disabled)
   Active: active (running) since Fri 2024-06-14 11:43:44 CST; 54s
ago
...
```

3.3.5.2 在 LSF 中启用模型预测服务（required）

可以使用 esub 的方式， 在 LSF 中使用该内存预测服务。

首先，将 memPrediction 安装目录下的 tools/esub.mem_predict 文件拷贝到 LSF 安装目录下的 10.1/linux3.10-glibc2.17-x86_64/etc 下面。（同 esub.default 目录，不同版本的目录名会有一些区别）

```
Bash
[root@openlava-master etc]# cp
/ic/software/tools/memPrediction/tools/esub.mem_predict .
```

然后采用 bsub -a mem_predict 的方式验证一下内存预测功能是否生效。

```
Bash
[root@openlava-master etc]# bsub -a mem_predict -q test ls
memPrediction: The recommended rusage memory value is: 296673MB.
Job <35746> is submitted to queue <test>.
```

如果生效，可以正式启用 esub.mem_predict 为默认的 esub 设置，在 LSF 安装目录下的 conf/lsf.conf 配置文件中增加如下行。

```
Bash
LSB_ESUB_METHOD="mem_predict"
```

最后，通过如下指令让配置生效。（root 或者 LSF 管理员账号操作）

```
Bash
lsadmin reconfig
```

该 esub.mem_predict 文件主要实现的是当预留内存为 0 的时候， 帮助用户使用模型预测的 memory 值进行内存预留的功能。

3.3.6 Web 应用部署 （optional）

memPrediction 提供 Web 应用的部署工具 setup，可以在 memPrediction 安装路径下的 web_app 文件夹下找到这个工具。该工具不需要额外配置即可在命令行端快速启动 Web 应用。

3.3.6.1 部署 Elastic Search 数据库

memPrediction 的 Web 应用使用的数据库是 Elastic Search 数据库， 因此在启动 Web 应用之前， 需要先启动 Elastic Search 数据库。

memPrediction 可以通过 `setup --db` 方式启动 Elastic Search 数据库， 需要注意的是，当前机器可以访问互联网， 是直接启动 Elastic Search 数据库的前提。

```
Bash
[zhangjingwen.silvia@n232-134-066:memPrediction]$ cd web_app/
[zhangjingwen.silvia@n232-134-066:web_app]$ bash setup --db
...
[2025-03-27 20:29:35] *Info*: Elasticsearch has started
successfully.
[2025-03-27 20:29:35] *Info*: Reset Password for Elastic Search
[2025-03-27 20:29:42] *Info*: Reset Password
[2025-03-27 20:29:42] *Info*: Rewrite configuration.
[2025-03-27 20:29:42] *Info*: Rewrite configuration done.
```

如果当前机器不能访问互联网， 那么需要先在联网机器中， 下载 Elastic Search 的 8.17.0 安装包:

```
Bash
curl -L -O
https://artifacts.elastic.co/downloads/elasticsearch/elasticsearch
-8.17.0-linux-x86_64.tar.gz
```

安装包下载完毕后， 需要将安装包 elasticsearch-8.17.0-linux-x86_64.tar.gz 传到离线机器上， 执行以下命令:

```Bash
[zhangjingwen.silvia@n232-134-066:web_app]$ bash setup --db --es
elasticsearch-8.17.0-linux-x86_64.tar.gz
...
[2025-03-27 20:29:35] *Info*: Elasticsearch has started
successfully.
[2025-03-27 20:29:35] *Info*: Reset Password for Elastic Search
[2025-03-27 20:29:42] *Info*: Reset Password
[2025-03-27 20:29:42] *Info*: Rewrite configuration.
[2025-03-27 20:29:42] *Info*: Rewrite configuration done.
```

3.3.6.2 部署 Web 后端服务

memPrediction 可以使用部署工具 setup， 执行 `setup --backend`， 直接启动 Web 后端应用。

```sql
[zhangjingwen.silvia@n232-134-066:web_app]$ bash setup --backend
[2025-03-27 20:32:16] *Info*: Starting the backend Flask
server ...
[2025-03-27 20:32:16] *Info*: Starting the backend Flask server
done.
[2025-03-27 20:32:16] *Info*: Rewrite configuration
[2025-03-27 20:32:16] *Info*: Rewrite configuration.
[2025-03-27 20:32:16] *Info*: Rewrite configuration done.
 * Serving Flask app 'web_app.backend.memory_web'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a
production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://0.0.0.0:12345
 * Running on http://10.232.134.66:12345
```

3.3.6.3 部署 Web 前端服务

memPrediction 可以使用部署工具 setup， 执行 `setup --frontend`， 启动最终可以进行信息检索和查看的 Web 前端应用。 在终端最终打印出的 Network 网址， 例如下面例子中的 `http://10.232.134.66:3000/`， 可以直接通过浏览器进行访问。

需要注意的是，当前机器上有 node 环境， 且可以访问互联网， 是直接启动前端服务的前提。此外， frontend 文件夹下的所有文件不能是链接文件， 否则会导致构建失败。

```sql
[zhangjingwen.silvia@n232-134-066:web_app]$ bash setup --frontend
[2025-03-27 20:35:06] *Info*: Starting the frontend Flask
server ...
> mem_web@1.0.0 build
> rsbuild build

...

> mem_web@1.0.0 preview
> rsbuild preview

  Rsbuild v0.6.15

  > Local:    http://localhost:3000/
  > Network:  http://10.232.134.66:3000/
```

如果当前机器没有 node 环境， 需要先安装 node 环境。

如果当前机器不能联网， 那么需要先在联网机器上， 使用和离线机器相同版本的 node 包，进入 memPrediction 的 web_app/frontend 目录， 执行以下命令：

```sql
npm install --legacy-peer-deps
```

执行结束后， 在安装包中会多一个名叫 node_modules 的文件夹， 需要将这个 node_modules 文件夹复制到不离线机器上， 然后执行如下命令：

```sql
[zhangjingwen.silvia@n232-134-066:web_app]$ bash setup --frontend
--node node_modules
[2025-03-27 20:35:06] *Info*: Starting the frontend Flask
```

```
server ...
> mem_web@1.0.0 build
> rsbuild build

...

> mem_web@1.0.0 preview
> rsbuild preview

  Rsbuild v0.6.15

  > Local:    http://localhost:3000/
  > Network:  http://10.232.134.66:3000/
```

3.3.6.4 数据库导入

经过以上步骤， 现在 Web 页面已经全部导入了， 但是因为数据库中还没有数据， 所以还不能看到信息。

memPrediction 的 Web 应用自带一个数据归档工具， 只需要将数据归档工具部署为一个定时任务， 就可以将 memPrediction 已经采集到的 job 数据，或者 lsfMonitor 采样到的 job 数据自动归档到数据库中。

该定时任务的 crontab 不需要额外配置环境变量，仅配置定时任务即可： `59 59,23 * * *${INSTALL_PATH}/web_app/setup --collect`，需要将其中的 ${INSTALL_PATH} 更改为安装路径。

以下是一个 crontab 的范例：

```sql
# For memPrediction web data Collect
59 59,23 * * * /ic/software/tools/memPrediction/web_app/setup --
collect
```

3.4 工具效果演示

在添加了该内存预测服务的 LSF 系统中提交 job 时， 如果用户没有设置内存， 会自动帮助用户进行内存预测， 并提示用户该 job 的内存预留量：

```Bash
[root@openlava-master etc]# bsub -q test ls
```

```
memPrediction: The recommended rusage memory value is: 120768MB.
Job <35747> is submitted to queue <test>.
```

可以看到， memPrediction 提供的内存预测服务预测该 job 的实际内存用量为 45MB，并会为用户的该 job 自动预留内存， 可以使用 `bjobs -UF <jobid>` 查看是否为确实为该 job 预留了内存。

```
Bash
[root@openlava-master etc]# bjobs -UF 35747

Job <35747>, User <root>, Project <default>, Service Class
<mysla>, Status <PEND>, Queue <test>, Command <ls>, Job
Description <ALLOC_MEMORY_USER=memPrediction(reset=120768MB)>,
Esub <mem_predict>
Fri Jun 14 13:48:43: Submitted from host <openlava-master>, CWD
</ic/software/tools/lsf_test/10.1/linux3.10-glibc2.17-x86_64/etc>,
Requested Resources < rusage[mem=120768]>;
 PENDING REASONS:
 Job requirements for reserving resource (mem) not satisfied: 2
hosts;


 SCHEDULING PARAMETERS:
           r15s   r1m  r15m   ut      pg    io   ls    it    tmp
swp    mem
 loadSched   -     -     -     -       -     -    -     -     -
-      -
 loadStop    -     -     -     -       -     -    -     -     -
-      -

 RESOURCE REQUIREMENT DETAILS:
 Combined: select[type == local] order[r15s:pg]
rusage[mem=120768.00]
 Effective: -
```

可以看到， 在查询结果中， 虽然用户没有为这个 job 预留内存， 但是在提交上去的 job 中， 该 job 的内存预留值为 45MB。

四、 工具介绍

memPrediction 包含四个不同的工具， 如下:

- **数据采集工具 sample**: 用于采集 lsf job 的信息， 数据将被按天分割保存， 用于数据的分析和训练。
- **数据分析工具 report**: 分析 lsf job 的信息， 包括 lsf 总体的内存预留情况（提交 job 时的 `-R "rusage[mem=$MEM]"`）, cpu 预留情况（提交 job 时的 `-n $SLOTS`）。
- **模型训练工具 train**: 通过机器学习的方法，用 lsf job 的数据训练一个 memory 模型，用于预测一个 job 的 max memory。
- **内存预测工具 predict**: 使用训练好的 memory 模型， 对新提交的 job 进行 max memory 的预测。

4.1 sample: 数据采集工具

memPrediction 提供采样工具 sample，sample 工具可以将 LSF 的 job 信息按天采集为一个 csv 文件。sample 位于 memPrediction 安装目录下的 bin/sample，安装后可以直接引用。如果使用环境中配置了 modules，则可以通过 module load 的方式引用 sample。

注：

memPrediction 兼容 lsfMonitor 中的 bsample 工具进行的 LSF job 信息采样，如果已经使用了 bsample -j 进行 job 信息的采样， 那么可以直接在配置文件 config.py 中修改如下两个值， 从而直接使用 lsfMonitor 采样的信息， 就不需要使用 memPrediction 自带的采样工具了。

- **db_path**: 修改为 bsample 采样的 job 信息的路径
- **job_format**: 默认为 csv，如果要使用 bsample 工具采样的信息， 这里可能需要修改为 'json' / 'sqlite'
  - 如果 lsfMonitor 采样到的数据是 json 格式， 那么需要将可以将 job_format 修改为 'json'

```
sql
...
# job infomation database save directory, format:
csv/sqlite.
db_path =
```

```
"/ic/software/cad_data/it/lsfMonitor/db/IC_CLUSTER/job"

# Specify job database format
job_format = 'json'
...
```

- 如果 lsfMonitor 采样到的数据保存到的 sqlite 数据库中， 那么需要将 job_format 修改为 'sqlite'

```
sql
...
# job infomation database save directory, format:
csv/sqlite.
db_path =
"/ic/software/cad_data/it/lsfMonitor/db/IC_CLUSTER/job"

# Specify job database format
job_format = 'sqlite'
...
```

4.1.1 帮助信息

sample 的帮助信息如下。

```
Bash
[root@ic-admin2 memPrediction]# bin/sample -h
usage: sample.py [-h] [-c] [-d]

optional arguments:
  -h, --help  show this help message and exit
  -c, --csv   Sample done job info and save as csv file
  -d, --db    Sample done job info and save as sqlite
```

- **csv（-c）**：以 csv 格式去保存收集到的 lsf 的信息， 推荐用这种方法去收集。
- **db（-d）**：以 sqlite 数据库去保存收集到的 lsf 的信息

4.1.2 采样范例

下面给到一个 sample 的采样范例，这次采样数据将以 csv 格式保存。(请注意，LSF 默认是不允许 root 账号使用的，需要配置才能放开权限)

```
Bash
[root@ic-admin2 memPrediction]# ./bin/sample -c
[2024-06-14 10:06:58] *Info*: >>> Sampling job info ...
[2024-06-14 10:09:13] *Info*:     Done ( 82917 jobs).
sampling_csv cost time: 135.64 s
```

采集到的数据会按天保存到 db_path 下。

```
Bash
[root@ic-admin2 memPrediction]# ls db/job_db/
job_info_20240614.csv
```

4.1.3 定时采样

我们推荐用 crontab 来定时采样（ Jenkins 类似），定时任务实际上在每天 23：59 采样一次即可（抓取当天 finished jobs），为防止意外我们在 22:59 多采样一次。

下面是一个示例。

```
Bash
[root@ic-admin2 memPrediction]# crontab -l
SHELL=/bin/bash
PATH=/ic/software/tools/python3/3.12.12/bin:/ic/software/tools/lsf
/10.1/linux3.10-glibc2.17-
x86_64/etc:/ic/software/tools/lsf/10.1/linux3.10-glibc2.17-
x86_64/bin:/bin:/sbin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/us
r/bin
LSF_SERVERDIR=/ic/software/tools/lsf/10.1/linux3.10-glibc2.17-
x86_64/etc
LSF_LIBDIR=/ic/software/tools/lsf/10.1/linux3.10-glibc2.17-
x86_64/lib
LSF_BINDIR=/ic/software/tools/lsf/10.1/linux3.10-glibc2.17-
x86_64/bin
LSF_ENVDIR=/ic/software/tools/lsf/conf

# For memPrediction, sample job info.
```

```
59 22,23 * * * /ic/software/tools/memPrediction/bin/sample -c
```

请注意，crontab 中默认是没有任何环境的，所以需要在 crontab 中设置好 PATH 等变量，否则 sample 中引用的 bjobs 等工具无法生效。这些变量可以通过如下方式获取。

```
Bash
[root@ic-admin2 memPrediction]# echo $SHELL
/bin/bash
[root@ic-admin2 memPrediction]# echo $PATH
/ic/software/tools/python3/3.12.12/bin:/ic/software/tools/lsf/10.1
/linux3.10-glibc2.17-
x86_64/etc:/ic/software/tools/lsf/10.1/linux3.10-glibc2.17-
x86_64/bin:/bin:/sbin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/us
r/bin
[root@ic-admin2 memPrediction]# env | grep LSF
LSF_SERVERDIR=/ic/software/tools/lsf/10.1/linux3.10-glibc2.17-
x86_64/etc
LSF_LIBDIR=/ic/software/tools/lsf/10.1/linux3.10-glibc2.17-
x86_64/lib
LSF_BINDIR=/ic/software/tools/lsf/10.1/linux3.10-glibc2.17-
x86_64/bin
LSF_ENVDIR=/ic/software/tools/lsf/conf
```

4.1.4 采样数据库

在 config/config.py 配置的 db_path 下面， 可以找到该次采样得到的数据， 该路径默认在安装路径下的 db/job_db 文件夹下。

采样得到的数据将保存为 **$YYYYmmdd.csv** 文件， 例如今天是 2024 年 6 月 14 日， 那么采集到的数据将保存在 job_info_20240614.csv 文件中。 如果是今天的首次采样， 将创建一个新的文件去保存信息； 如果该文件已经存在， 那么新采样的数据在去重后， 将被追加到该文件中。

采样数据采样的是， 今天到采样时间点为止最终状态为 "DONE"，"EXIT" 的 job。采集的信息为该 job 的下列表格中的信息。

Sampling item	Description
job_id	lsf job 的 job id， 用于检查是否收集到了相同的 job

started_time	lsf job 开始的时间
job_name	job 在提交的时候通过 -J 指定的 job name
user	job 的 user
status	job 在结束时候的状态，采集的 job 中为"DONE" 或者 "EXIT"
project	job 在提交时候指定的 -P，为该 job 指定的项目
cwd	job 在提交时候的路径，例如在家目录下提交， 目录为 ~ 的路径
command	job 具体的 command，例如 sleep 10，innovus -stylus 等
finished_time	job 结束的时间， 因为只采集 DONE，EXIT 的 job， 一定会存在一个正常完成或者异常退出的时间
processors_requested	job 在提交时候指定的 -n，为用户给 job 指定的 slots 数量， 如果不指定的话， 这里是 1
interactive_mode	job 在提交是时候是否使用了 interactive 模式， 如果使用了 interactive 模式的话为 True， 如果不是的话为 False
cpu_time	job 一共使用了多少 cpu time，单位为 s
span_hosts	job 在提交时候 -R 指定的 "[span_hosts=$span_hosts]" 数量
job_description	job 在提交时候指定的 -Jd，也就是对该 job 的描述
max_mem	job 在运行过程中使用的最大内存，单位与 lsf 中对 memory 单位的设置相同， 也是模型预测的目标值
avg_mem	job 在运行过程中使用的平均内存，单位与 max memory 相同
rusage_mem	在提交 job 的时候指定 -R "[rusage=$rusage_mem]" 的值， 为用户为该 job 预留的 memory

4.2 report: 数据分析工具

report 位于 memPrediction 安装目录下的 bin/report，安装后可以直接引用。如果使用环境中配置了 modules，则可以通过 module load 的方式引用 report。

数据分析支持对 **memory** 预留情况和 **cpu** 预留情况 的分析。

注意， 数据分析的前提是， 在想要分析的时间段中， 已经采用数据采样的工具进行了数据采集。在采集到的数据基础上， 才可以进行进一步的数据分析。

4.2.1 帮助信息

```
Bash
[root@ic-admin2 memPrediction]# bin/report -h
usage: report.py [-h] [-st START_DATE] [-et END_DATE] [-db DB] [-
m] [-c]

optional arguments:
  -h, --help            show this help message and exit

rusage rpt csv:
  -st START_DATE, --start_date START_DATE
                        analysis rpt from start date, format:
YYYY-mm-dd, default 30days ago
  -et END_DATE, --end_date END_DATE
                        analysis rpt to end date, format: YYYY-mm-
dd, default today
  -db DB                directory path including all job data
  -m, --memory          memory analysis
  -c, --cpu             cpu analysis
```

数据分析来自于数据库中采集的 job 信息。 可以选定想要的分析时间段，将最早的一天定义为 start date，将最晚的一天定义为 end date， 可以在命令行按照如下方式指定:

- **start_time（-st）**：数据的开始日期， 不指定的话默认为即时起的三十天之前， 需要用 YYYY-mm-dd 的方式去指定
- **end_date（-et）**：训练数据的结束日期， 不指定的话默认为当天， 需要用 YYYY-mm-dd 的方式去指定结束日期
- **db**: 存放采集到的 job 信息的路径， 如果不定义的话， 默认为在 config 中定义的 db_path。
- **memory（-m）**：对该时间段内的 job memory 预留情况进行分析

- **cpu（-c）**：对该时间段内的 job cpu 预留情况进行分析

4.2.2 memory 分析

在不指定时间和数据库参数的情况下，通过-m 参数可以获取最近 30 天的 memory 分析报告。

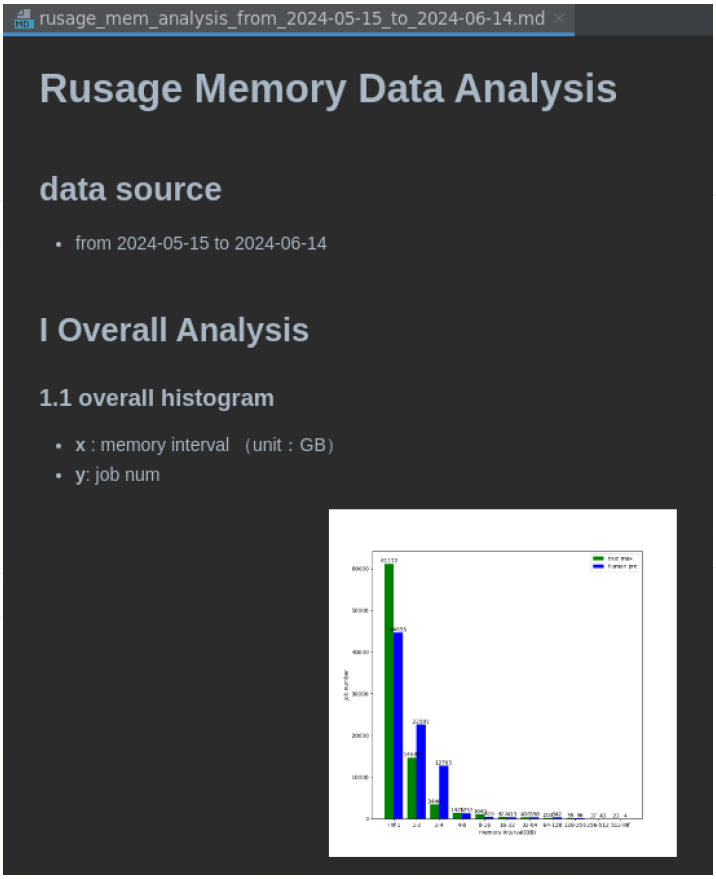
```
Bash
[root@ic-admin2 memPrediction]# bin/report -m
...
[2024-06-14 11:07:36] *Info*: Report Path:
rusage_mem_analysis_from_2024-05-15_to_2024-06-14.md
```

生成的数据如下。

```
Bash
[root@ic-admin2 memPrediction]# ls db/report_db/
pictures_memory  rusage_mem_analysis_from_2024-05-15_to_2024-06-
14.md  tables_memory
```

4.2.2.1 memory 分析报告

报告默认生成在 db/report_db 下面，格式为 markdown，样式如下。

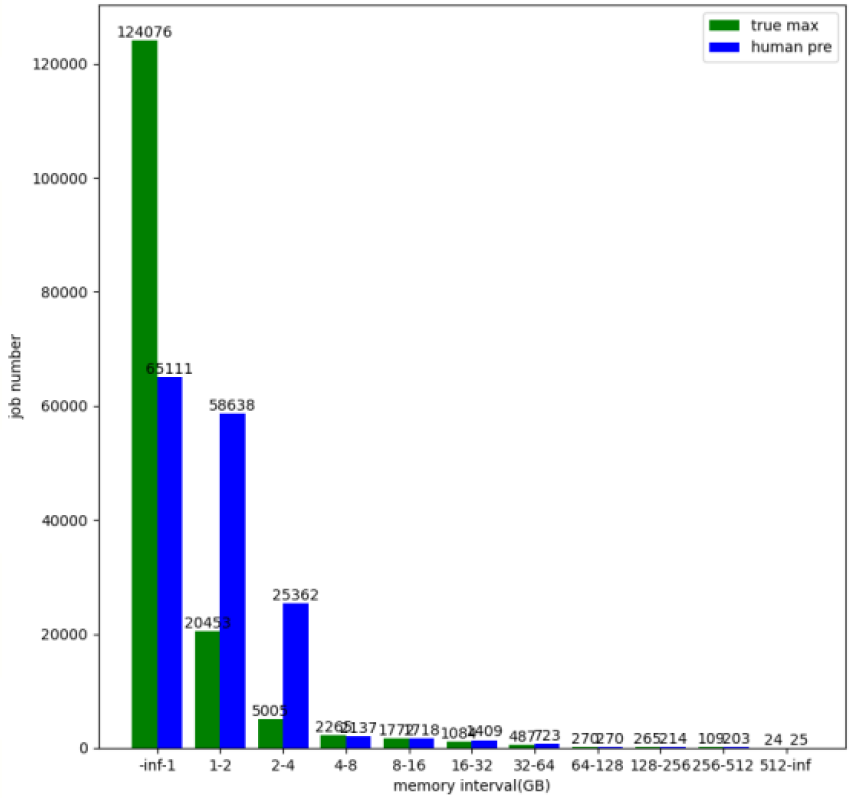


该报告使用工具默认的 memory 分析报告模板， 分析报告模板可以在 config 中的 report_template 指定。

4.4.2.2 memory 分析图表

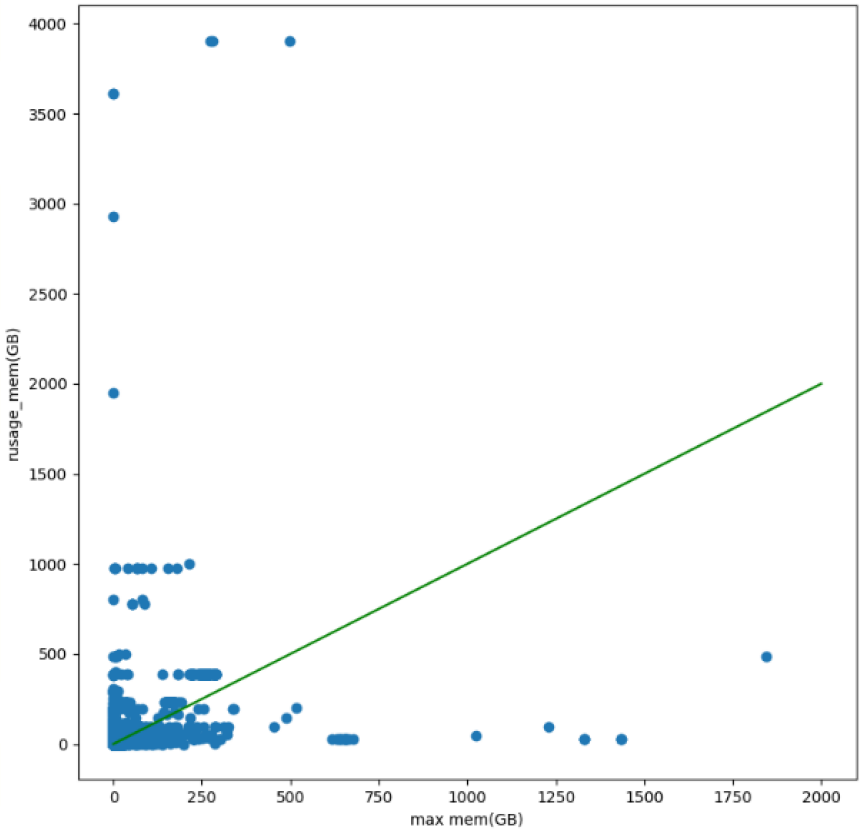
- **overall histogram**

整体 job 的 memory 预留情况条形图， 展示了在各个区间内 job 真实的 max memory 分布情况和用户预留的 rusage memory 分布情况。



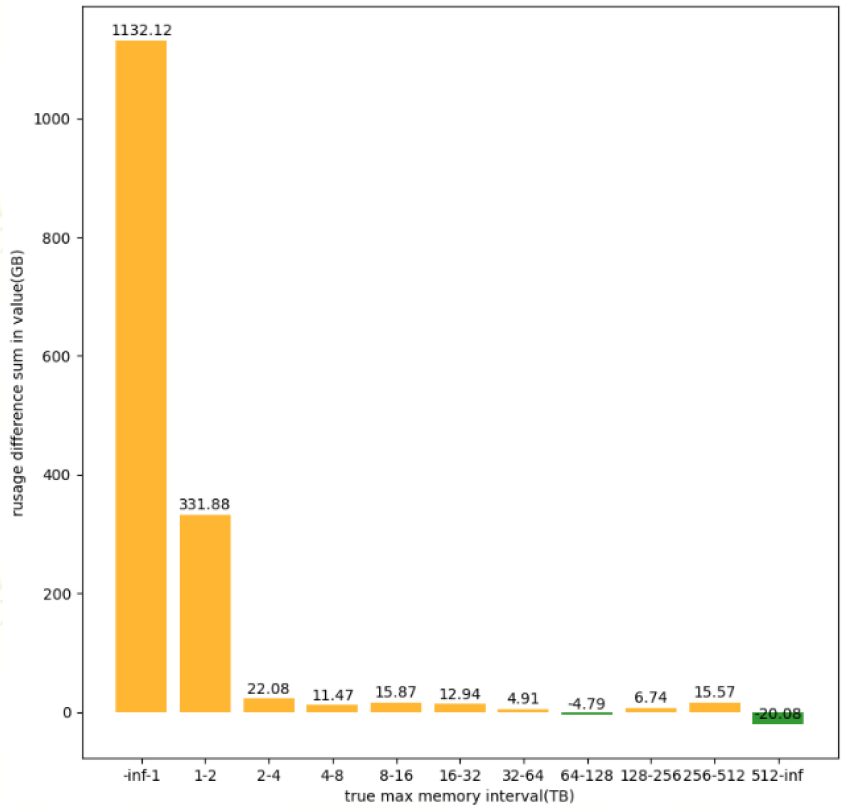
- **overall scatter**

整体 job 的 memory 预留情况散点图， 展示了在各个 job 真实的 max memory 分布情况和用户预留的 rusage memory 总体分布情况。



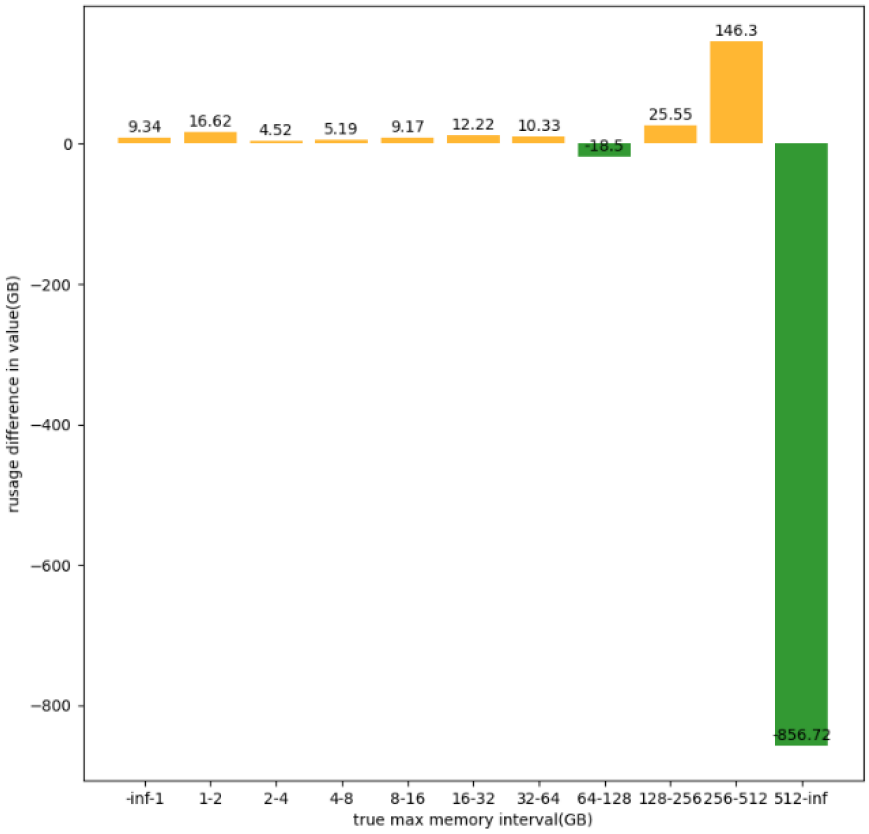
- **rusage difference sum**

  job 的 max memory 和对应的用户预留的 rusage memory 差值在各个区间的和。也就是每个区间内的 job 预留多了的 memory 的总和。



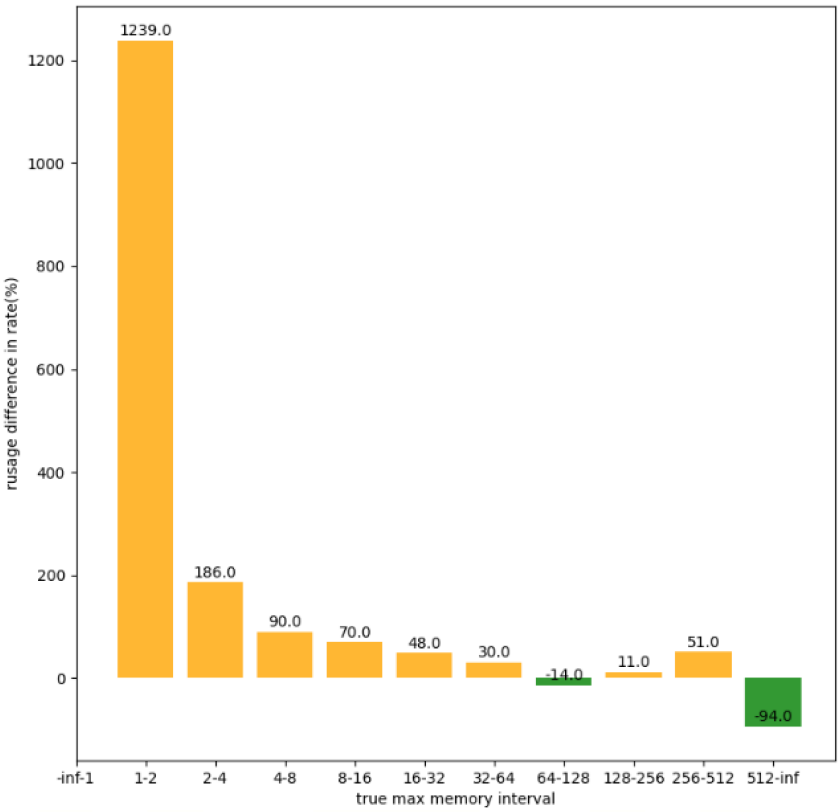
- **rusage difference in value**

各个区间内 job 的（**用户预留 rusage memory - job 真实 max memory**）的平均值， 反映每个区间预留的平均误差。



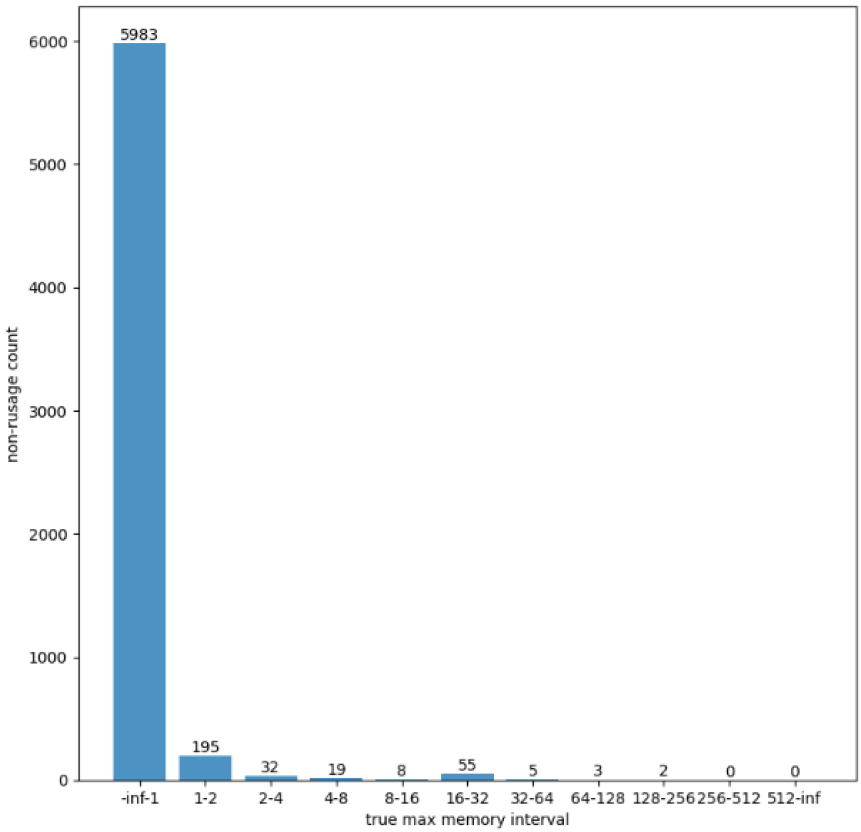
- **rusage difference in rate**

各个区间内 job 的**（用户预留 rusage memory - job 真实 max memory）/ job 真实 max memory * 100 %** 的平均值， 反映每个区间预留的平均误差率。



- **non rusage memory analysis**

  在各个区间内， job 没有预留 memory 的数量。



- **tolerance pie chat**

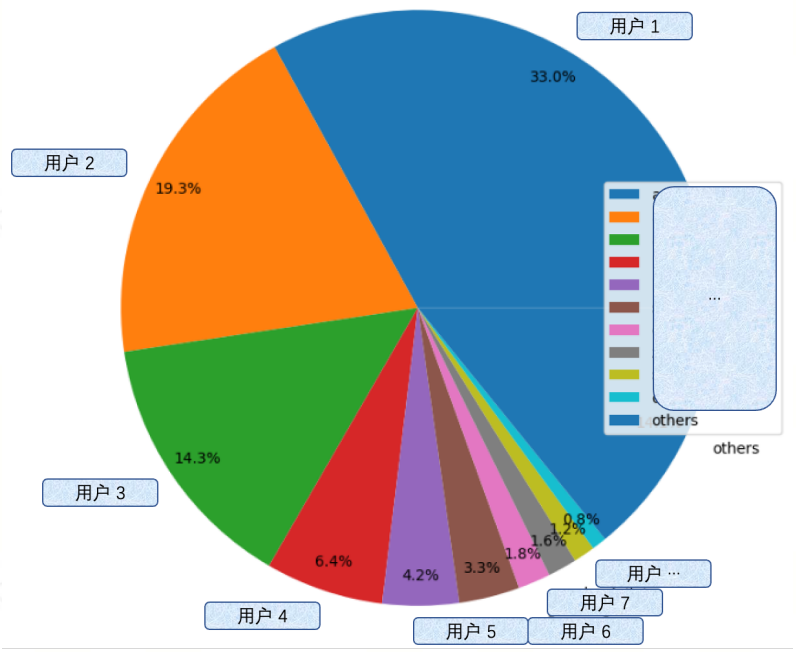
  首先定义用户预留 memory 的容许范围。

  - **容许范围外多预留 memory**： 使用 **rusage memory - max memory * （1 + boundary）** 计算该 job 多预留 memory

  - 容许比例表：**boundray_list** = [4, 3, 2, 1, 1, 0.5, 0.5, 0.3, 0.3, 0.3, 0.3]

    - 解释:
    - 0-1: 容许 rusage_mem 指定为 max mem*5
    - 1-2: 容许 rusage_mem 指定为 max mem*4
    - 2-4: 容许 rusage_mem 指定为 max mem*3
    - 4-16: 容许 rusage_mem 指定为 max mem*2
    - 16-64: 容许 rusage_mem 指定为 max mem*1.5
    - 64+: 容许 rusage_mem 指定为 max mem*1.3

  选择多预留情况最多的 10 位用户， 展示多预留 memory 总量的占比。



4.4.3 cpu 分析

在不指定时间和数据库参数的情况下，通过-c 参数可以获取最近 30 天的 cpu 分析报告。

```
Bash
[root@ic-admin2 memPrediction]# bin/report -c
...
[2024-06-14 11:09:59] *Info*: Data process done.
```

生成的数据如下（包含 memory 数据）。

```
Bash
[root@ic-admin2 memPrediction]# ls db/report_db/
pictures_cpu  pictures_memory  rusage_mem_analysis_from_2024-05-
15_to_2024-06-14.md  tables_cpu  tables_memory
```

cpu 分析无法提供典型的分析项内容，因此不会生成报告，但是会生成一些统计图表备查，会生成在 db/report_db 下的 pictures_cpu 和 tables_cpu 目录中。

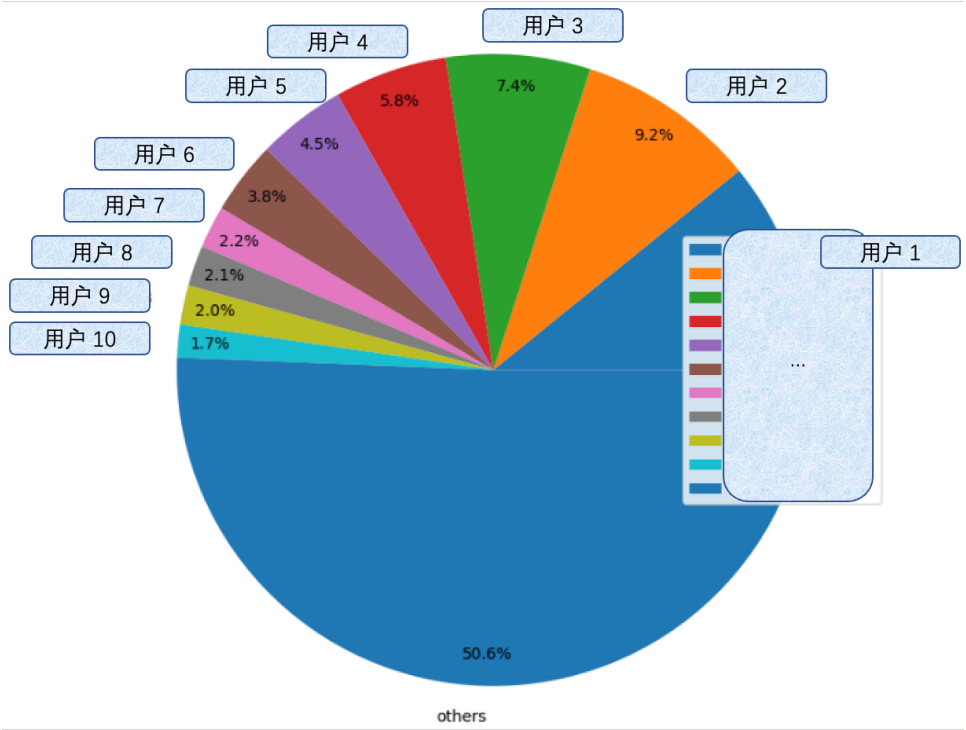
4.4.3.1 cpu 分析图表

- **over processors requested user pie**

  先解释 cpu 预留的计算方法：

  - **requested processors**: 预留的 cpu, 用户在提交 job 的时候 -n 指定的参数
  - **used processors**：实际使用的 cpu, 用户 job 正常结束后，**cpu time / run time = used processors**
  - **cpu utilization**: cpu 利用率，**used processors / requested processors = cpu utilization**
  - **多预留的 cpu**：requtested processors - used processors 的计算结果作为多预留的 cpu

选择多预留情况最多的十位用户，展示多预留 cpu 数量的占比



4.3 train： 模型训练工具

train 位于 memPrediction 安装目录下的 bin/train，安装后可以直接引用。如果使用环境中配置了 modules，则可以通过 module load 的方式引用 train。

4.3.1 帮助信息

```
Bash
[root@ic-admin2 memPrediction]# bin/train -h
usage: train.py [-h] [-st START_DATE] [-et END_DATE] [-db
DATA_PATH] [--training_cfg TRAINING_CFG]

optional arguments:
  -h, --help            show this help message and exit

training model:
  -st START_DATE, --start_date START_DATE
                        analysis rpt from start date, format:
YYYY-mm-dd, default 60 days ago
  -et END_DATE, --end_date END_DATE
                        analysis rpt to end date, format: YYYY-mm-
dd, default today
  -db DATA_PATH, --data_path DATA_PATH
                        directory path including all training data
  --training_cfg TRAINING_CFG
                        config for training model parameter,
format: yaml
```

训练数据来自于数据库中采集的 job 信息。由于采集数据是按天分割的， 所以需要指定取数据库中的哪天到哪天的数据作训练数据。将最早的一天定义为 start date，将最晚的一天定义为 end date， 可以在命令行按照如下方式指定：

- **start_time（-st）**：数据的开始日期， 不指定的话默认为即时起的三十天之前， 需要用 YYYY-mm-dd 的方式去指定
- **end_date（-et）**：训练数据的结束日期， 不指定的话默认为当天， 需要用 YYYY-mm-dd 的方式去指定结束日期
- **db**: 存放采集到的 job 信息的路径， 如果不定义的话， 默认为在 config 中定义的 db_path。

- **--training_cfg**： 模型训练的 config 文件路径， 如果不指定的话默认使用工具自带的 config 进行训练。

4.3.2 模型训练

模型训练支持使用指定数据集， 指定数据集的指定时间段进行训练， 并且可以通过更改模型训练的 config 来调整模型训练的参数。

4.3.2.1 默认参数

使用 config 定义的 db_path 中， 从 60 天前到现在的数据进行训练。

```
Bash
[root@ic-admin2 memPrediction]# bin/train
...
```

4.3.2.2 指定时间段

使用 config 中定义的 db_path，选择 2024 年 4 月 1 日到 2024 年 4 月 30 日的数据进行训练。

```
Bash
[root@ic-admin2 memPrediction]# bin/train -st 2024-04-01 -et 2024-
04-30
```

4.3.2.3 指定数据路径

使用自定义路径 `$DB_PATH` 中的数据进行训练。

```
Bash
[root@ic-admin2 memPrediction]# bin/train -db $DB_PATH
```

4.3.2.4 指定模型训练 config

使用自定义的模型 config 文件 `$CONFIG_PATH` 进行训练。

```
Bash
[root@ic-admin2 memPrediction]# bin/train --training_cfg
$CONFIG_PATH
```

4.3.3 模型训练结果

模型训练结束后会打印该模型训练用时， 该模型的 MSE 等。

```
Bash
[2024-06-14 11:16:54] *Info*: Training mse is
2.692679581837349 ...
[2024-06-14 11:16:54] *Info*: max mem interval value counts is
 max_mem_interval
(-inf, 1.0]       570
(1.0, 2.0]        152
(2.0, 4.0]         26
(4.0, 8.0]         14
(8.0, 16.0]        12
(16.0, 32.0]        4
(32.0, 64.0]        3
(64.0, 128.0]       2
(128.0, 256.0]      0
(256.0, 512.0]      0
(512.0, inf]        0
Name: count, dtype: int64
main cost time: 188.43 s
```

模型的训练结果会存放在 config 中定义的 **model_db_path** 中， 文件夹名称为模型训练的开始时间点。例如从 2024 年 6 月 14 日 11 点 13 分开始训练， 就会在 model_db_path 下生成一个 2024_06_14_11_13 的文件夹， 里面存放了模型训练的结果。

```
Bash
[root@ic-admin2 memPrediction]# ls db/model_db/2024_06_14_11_13/
cat_encs  config  model  rpt
```

模型文件中包含四个部分， 分别为 **cat_encs**，**config**，**model**，**rpt**。这些文件都是模型训练完成后自动生成的。

4.3.3.1 cat_encs

cat_encs 文件夹下面存放了模型在训练之前， 将数据进行编码的编码文件。

在预测中要使用同样的编码映射表进行编码， 因此在这个文件夹下存放了原始映射表的二进制文件， 在使用这个模型进行预测的时候， 会将这个文件 load 到脚本中。

```
Bash
[root@ic-admin2 memPrediction]# ls
db/model_db/2024_06_14_11_13/cat_encs/
cat_encs.file
```

4.3.3.2 config

config 文件夹下存放了 config 文件， 定义了该模型的相关参数， 包括这个模型训练使用的参数， 模型训练集的因子， 模型和子模型文件的路径， 模型训练集的规模等。在 memory 的预测中， config 定义了预测的基本行为和调用模型时候模型的文件路径。

```
Bash
[root@ic-admin2 memPrediction]# ls
db/model_db/2024_06_14_11_13/config/
config
```

4.3.3.3 model

model 文件夹中存放了 memory 预测模型的二进制文件， 以及该模型使用的子模型的二进制文件。在使用这个模型进行预测的时候， 会将这些模型文件 load 到脚本中。

```
Bash
[root@ic-admin2 memPrediction]# ls
db/model_db/2024_06_14_11_13/model/
command_glove.corpus  command_word2vec.model  cwd_glove.model
job_name_glove.corpus  job_name_word2vec.model  xgb_reg.model
command_glove.model   cwd_glove.corpus        cwd_word2vec.model
job_name_glove.model   user_df.dic
```

4.3.3.4 rpt

rpt 文件夹中， 存放了一个关于模型的 rpt 文件。

```Bash
[root@ic-admin2 memPrediction]# ls
db/model_db/2024_06_14_11_13/rpt/
rpt
```

该文件中存放了模型评估的结果， 主要包括：

- **RMSE** （均方误差）
- **memory 分段的平均误差和平均误差率**

一个 rpt 的示例如下，一般会使用该文件作为模型评价的标准， 选择结果较好的模型用于预测：

```YAML
The RMSE is 2.692679581837349
+--------------------+---------------+----------------+-----------
----------+
| max_mem_interval   |   max_mem_num |   pre_mem_diff |
pre_mem_diff_rate |
|--------------------+---------------+----------------+-----------
----------|
| (-inf, 1.0]        |           570 |       0.489297 |
nan       |
| (1.0, 2.0]         |           152 |      -0.214242 |
-13.8     |
| (2.0, 4.0]         |            26 |      -1.19787  |
-40.8868  |
| (4.0, 8.0]         |            14 |       1.34473  |
9.86984 |
| (8.0, 16.0]        |            12 |      -5.92496  |
-52.8352  |
| (16.0, 32.0]       |             4 |     -13.4383   |
-70.4508  |
| (32.0, 64.0]       |             3 |     -20.7077   |
-42.6228  |
| (64.0, 128.0]      |             2 |     -15.6559   |
-22.0368  |
| (128.0, 256.0]     |             0 |     nan        |
```

```
nan       |
| (256.0, 512.0]     |             0 |     nan        |
nan       |
| (512.0, inf]       |             0 |     nan        |
nan       |
+--------------------+---------------+----------------+-----------
----------+
```

在这个结果中， RMSE 指的是均方误差， 这个值越小， 代表模型越好；

下面的表格中， 第一列代表的是 job max memory 的区间；第二列代表测试集有多少数据落在这个区间； 第三列代表**平均误差**， 也就是该区间的 job max memory 的 （模型预测值 - 真实值） 的平均值；第四列指的是**平均误差率**， 也就是该区间的 （预测值 - 真实值）/ 真实值 * 100% 的平均数。

平均误差和平均误差率绝对值越小， 模型越好。

模型的选择需要综合以上三个数据进行考虑。

4.3.4 调整模型训练参数

memPrediction 中， config 下存放了一个默认的 training_config.yaml，用于定义模型训练的参数。其中使用的参数已经经过 fine-tuning，在大多数情况下能够得到较好结果。

但是如果有需要的话， 也可以修改其中的一些参数， 从而得到更符合当前情况的结果。

training_config.yaml 主要定义了如下参数。

- **报告参数**

```yaml
# 这里定义了模型生成的报告（rpt）中， 对 job 的 memory 预测情况进行评估时候的 memory（GB） 划分区间，可以通过增加或者删除的方式自由定义你想看到的报告 memory 区间样式。
# 这里代表会分别统计 (inf, 1], (1, 2], (2, 4], ..., (256, 512], (512, +inf) memory 区间内的 job 预测情况， 生成报告
report:
    bins:
        - -.inf
        - 1
        - 2
        - 4
        - 8
```

```
        - 16
        - 32
        - 64
        - 128
        - 256
        - 512
        - .inf
```

修改范例:

```
YAML
# 这里代表会分别统计 (inf, 16], (16, 32], (32, 64], (100, 200], (200, +inf) memory 区间内的 job 预测情况， 生成报告
report:
    bins:
        - -.inf
        - 16
        - 32
        - 64
        - 100
        - 200
        - .inf
```

- 交叉验证参数

```
YAML
# 这里定义了训练集和测试集的比例， test_size 是测试集所占的比例，train_size 是训练集所占的比例
# 这里代表了使用 99% 的数据作为训练集， 1% 的数据作为测试集， 数据不太充足的时候推荐这个设置
cross_validation:
    test_size: 0.01
    train_size: 0.99
```

修改范例:

```
YAML
# 这里代表了使用 90% 的数据作为训练集， 10% 的数据作为测试集， 数据充足的时候推荐这个设置
```

```
cross_validation:
    test_size: 0.1
    train_size: 0.9
```

- 子模型参数

```
YAML
# 定义了子模型的参数，此处支持 NPL 模型
# 默认对 job 的 cwd/command/job_name 信息都应用 Word2Vector 模型和 GloVe 模型进行向量化处理， 并对向量进行聚类处理
base_model:
    cwd: # 模型处理的变量： job 信息中的 cwd
        word2vec:  # 使用的模型， 支持 word2vec 和 glove，可以分开使用， 也可以一起使用
            emb_size: 128 # word2vec 生成的向量维数， 向量维数越多，生成越慢
#            cluster: 32 # 将生成的向量进行聚类的聚类中心数量， 如果不定义的话就不进行聚类
        glove:
            emb_size: 128
#            cluster: 32
    command: # 模型处理的变量： job 信息中的 command
        word2vec:
            emb_size: 128
#            cluster: 32
        glove:
            emb_size: 128
#            cluster: 32
    job_name: # 模型处理的变量： job 信息中的 job_name
         word2vec:
            emb_size: 128
#            cluster: 32
         glove:
            emb_size: 128
#            cluster: 32
```

修改范例:

```
YAML
# 对向量进行聚类处理， 并将聚类中心设置为 4
```

```
# 不对 job 的 job_name 信息做处理
base_model:
    cwd:
        word2vec:
            emb_size: 128
           cluster: 4
        glove:
            emb_size: 128
           cluster: 4
    command:
        word2vec:
            emb_size: 128
           cluster: 4
        glove:
            emb_size: 128
           cluster: 4
```

- 采样策略参数

```
YAML
# 定义采样策略参数
# 这部分定义了对训练模型的训练集进行的采样处理， 可以使得训练数据在各个区间分布更均衡， 有利于提高大内存 job 的预测精度
# 默认关闭
# Sampling:
#    max_mem: #以 max memory 作为采样的基准
#        status: True
#        bins: # 将 max memory 划分到不同的区间， 对区间进行采样
#            - -.inf
#            - 16
#            - 32
#            - 64
#            - 128
#            - .inf
#        over_sample: # 采用过采样方法
#            status: True
#            method: SMOTE #使用 SMOTE 过采样方法进行采样
#            sampling_strategy: auto # SMOTE 的采样策略
```

修改范例:

```yaml
# 取消掉注释则生效
Sampling:
    max_mem:
        status: True
        bins:
            - -.inf
            - 16
            - 64
            - 128
            - .inf
        over_sample:
            status: True
            method: SMOTE
            sampling_strategy: auto
```

- 模型训练参数

```yaml
# xgboost 的模型参数， 可以按需进行调节和增减
model:
    model_name: xgboost
    fitting_parameter:
        eval_metric: rmse
    training_parameter:
        max_depth: 15
        min_child_weight: 6
        n_estimators: 100
        objective: reg:tweedie
        seed: 64
        tweedie_variance_power: 1.2
```

修改范例:

```yaml
model:
    model_name: xgboost
    fitting_parameter:
        eval_metric: rmse
```

```
    training_parameter:
        max_depth: 15
        min_child_weight: 6
        n_estimators: 128
        objective: reg:tweedie
        seed: 64
        tweedie_variance_power: 1.3
```

4.3.5 预测模型配置

一个模型训练好了以后， 要将其用于 memory 的预测， 有两种模型配置方法。

4.3.5.1 config 配置

如果希望在后续的 memory 预测中， 都使用同一个模型进行预测， 可以在 config.py 中配置该模型为 predict_model，predict_model 默认指向一个 latest 路径，每次 training 的时候这个 latest 路径也会被重新指向最新的模型。

```Bash
# prediction model config yaml
predict_model =
"/ic/software/tools/memPrediction/db/model_db/latest"
```

正常情况下你都不需要修改这个设置，当然你也可以把它修改为你想使用的 model 模型路径。

4.3.5.2 predict 命令行配置

如果只是测试模型的预测效果， 可以直接在进行预测的时候， 通过命令行的方法指定使用的模型。详细方法会在下文的 "predict: 内存预测工具" 中介绍， **注意命令行中不可以直接指定模型文件夹， 而是要直接指定该模型的 config 文件**， 该文件为模型训练结束后自动生成的文件， 存放于模型文件夹中的 config 文件夹中， 也就是 config/config 文件。

4.4 predict: 内存预测工具

predict 位于 memPrediction 安装目录下的 bin/predict，安装后可以直接引用。如果使用环境中配置了 modules，则可以通过 module load 的方式引用 predict。

4.4.1 帮助信息

```
Bash
[root@ic-admin2 memPrediction]# bin/predict -h
usage: predict.py [-h] [--job_yaml JOB_YAML] [-c CONFIG] [--
job_name JOB_NAME] [--cwd CWD] [--command COMMAND] [--project
PROJECT] [--queue QUEUE] [--started_time STARTED_TIME]
                  [--rusage_mem RUSAGE_MEM] [--user USER] [-d]

optional arguments:
  -h, --help            show this help message and exit

predict job memory:
  --job_yaml JOB_YAML   job infomation in order to predict job max
memory(gb)
  -c CONFIG, --config CONFIG
                        predict model config
  --job_name JOB_NAME   job name
  --cwd CWD             job exec path
  --command COMMAND     job command
  --project PROJECT     job project
  --queue QUEUE         job queue
  --started_time STARTED_TIME
                        job begin time
  --rusage_mem RUSAGE_MEM
                        job reserve memory
  --user USER           job submission user
  -d, --debug           debug mode
```

- **config（-c）**：预测 memory 使用的模型
  - 不指定的话， 使用在 config 中配置的 predict_model 这里的模型
  - 可以通过指定模型的 config 文件指定模型， 例如使用 2023_12_29_17_01 这个模型，可以指定为 -c 2023_12_29_17_01/config/config

预测工具提供两种读取 job 信息的方法， 分别是命令行指定和 yaml 文件指定。

- **job_yaml**：常用于测试，存放了 job 的基本信息，格式为 yaml。一个 yaml 的示例如下：

```yaml
YAML
job_name: $JOB_YAML
user: $USER
project: $PROJECT
queue: $QUEUE
cwd: $CWD
command: $COMMAND
rusage_mem: $RUSGAE_MEM
started_time: $STARTED_TIME
```

参数含义如下：

item	Description
started_time	lsf job 开始的时间
job_name	job 在提交的时候通过 -J 指定的 job name
user	job 的 user
project	job 在提交时候指定的 -P，为该 job 指定的项目
cwd	job 在提交时候的路径，例如在家目录下提交，目录为 ~ 的路径
command	job 具体的 command，例如 sleep 10，innovus -stylus 等
rusage_mem	在提交 job 的时候指定 -R "[rusage=$rusage_mem]" 的值，为用户为该 job 预留的 memory

- **命令行**：常用于在 lsf 中对 job 读取 job 信息并进行预测，包括以下参数：

item	Description
--started_time	lsf job 开始的时间
--job_name	job 在提交的时候通过 -J 指定的 job name

--user	job 的 user
--project	job 在提交时候指定的 -P，为该 job 指定的项目
--cwd	job 在提交时候的路径，例如在家目录下提交， 目录为 ~ 的路径
--command	job 具体的 command，例如 sleep 10，innovus -stylus 等
--rusage_mem	在提交 job 的时候指定 -R "[rusage=$rusage_mem]" 的值， 为用户为该 job 预留的 memory

4.4.2 预测 memory

4.4.2.1 使用 job_yaml

如果已经按上文所述， 配置好了一个 job_yaml， 可以按照以下命令对该 job 进行内存预测。

```
Bash
[root@ic-admin2 memPrediction]# bin/predict --job_yaml $JOB_YAML
```

4.4.2.2 使用命令行

可以在命令行中传入 job 的相关信息。

```
Bash
[root@ic-admin2 memPrediction]# bin/predict --job_name $job_name
--user $user -- started_time $started_time --command $command --
cwd $cwd --project $project --queue $queue
```

4.4.2.3 指定预测模型

如果不想使用 config 中配置好的预测模型， 比如在测试一个新训练好的模型的时候， 可以使用 **-c 指定想使用的模型的 config 文件。**

```
Bash
```

```
[root@ic-admin2 memPrediction]# bin/predict -c $MODEL_CONFIG ...
```

4.4.3 预测结果

作为验证，我们将已完成的 job 41602169 的信息填到 41602169.yaml，其实际最大使用 memory 为 268GB，内存预测结果如下。

```
Bash
[root@ic-admin2 memPrediction]# bin/predict --job_yaml
41602169.yaml
...
[2024-06-14 11:36:48] *Info*: predict max memory is 277801 MB
predict cost time: 0.83 s
277801
```

预测结果为 277802 MB，即 271GB，跟实际使用内存量十分接近。

4.4.3.1 获取预测结果

如果要使用 bash 脚本， 获取预测到的 memory 的值， 可以使用如下方法：

```
Bash
# 定义预测工具的位置
PREDICT_SCRIPT='/ic/software/tools/memPrediction/bin/predict'

# 对 job 进行预测
JOB_PREDICT_MEMORY_CONTENT=$($PREDICT_SCRIPT --job_name  --
job_name $job_name
--user $user -- started_time $started_time --command $command --
cwd $cwd --project $project --queue $queue 2>&1)

#读取预测结果
JOB_PREDICT_MEMORY=$(echo "$JOB_PREDICT_MEMORY_CONTENT" | tail -
n1)
```

这里的 $JOB_PREDICT_MEMORY 就是该 job 预测得到的 max memory 大小。

4.5 内存预测服务

如果在 lsf 中， 直接使调用脚本的方式， 获取 memory 预测值，速度会比较慢。此时，可以使用脚本提供的工具， 使用 gunicorn + flask 的方式， 部署 memory 预测的 web 服务，加快预测速度。

4.5.1 web 服务搭建

首先， 安装工具后， 进入到安装路径下面的 tools 路径。

```
Bash
[root@ic-admin2 memPrediction]# cd tools/
[root@ic-admin2 tools]# ls
esub.mem_predict  predict_gconf.py  predict_web.py
predict_web.service  stopservice.sh  train.sh  update  update.py
web_startup.sh
```

可以看到， tools 文件夹下包含了几个主要的文件:

- **predict_gconf.py**: 定义了 web 服务的基本参数， **在安装时候会随机生成一个当前机器没有被占用的端口号。如果该端口后来被占用了， 或者需要部署其他服务， 务必切换至其他可用的端口。手动修改端口号后， 在 esub.mem_predict 脚本中也需要修改请求端口为对应的端口号。**

```
Python
import gevent.monkey
gevent.monkey.patch_all()

debug = True
workers = 5
worker_class = "gevent"
bind='0.0.0.0:$PORT' # 在这里修改端口号
```

- **predict_web.py**： 使用 flask 搭建的 memory 预测 API
- **predict_web.service**: web 服务的 service 服务文件， 用于起 memory 预测的 web 服务
- **stopservice.sh**: kill 掉该 web 服务的脚本

需要一台 linux 机器用于起 web 服务。记住该机器的 ip 地址， 调用 API 的时候需要使用该 ip。

- 将 predict_web.service 文件放入该系统中的 /usr/lib/systemd/system/目录下
- 启动服务

```Bash
systemctl start predict_web
```

- 停止服务

```Bash
systemctl stop predict_web
```

部署好了 web 服务后， 可以用如下方式去请求该 API， 得到模型预测值。

```Python
# 调用 memory 预测的 API
# 黄色部分为 启动 web 服务的机器的 ip 地址
# 绿色部分为 web 服务使用的端口号
JOB_PREDICT_MEMORY_CONTENT=$(curl http://$IP:$PORT/memPrediction -s -X POST --data-urlencode strated_time="$started_time" --data-urlencode job_name="$job_name" --data-urlencode command="$command" --data-urlencode cwd="$cwd" --data-urlencode user="$user" --data-urlencode queue="$queue" --data-urlencode project="$project")

# 判断 API 的请求是否成功
exit_code=$?

# 获取 memory 预测值
JOB_PREDICT_MEMORY=$(echo "$JOB_PREDICT_MEMORY_CONTENT" | tail -n1)
```

4.6 memory 分析 Web 应用

如果已经部署了 memPrediction 的 Web 应用， 就可以通过浏览器， 使用部署好的前端界面的 URL 链接， 直接访问到 memPrediction 的 Web 应用界面。如果还没有部署， 需要参考第三章 3.3.6 Web 应用部署， 进行部署， 再使用。

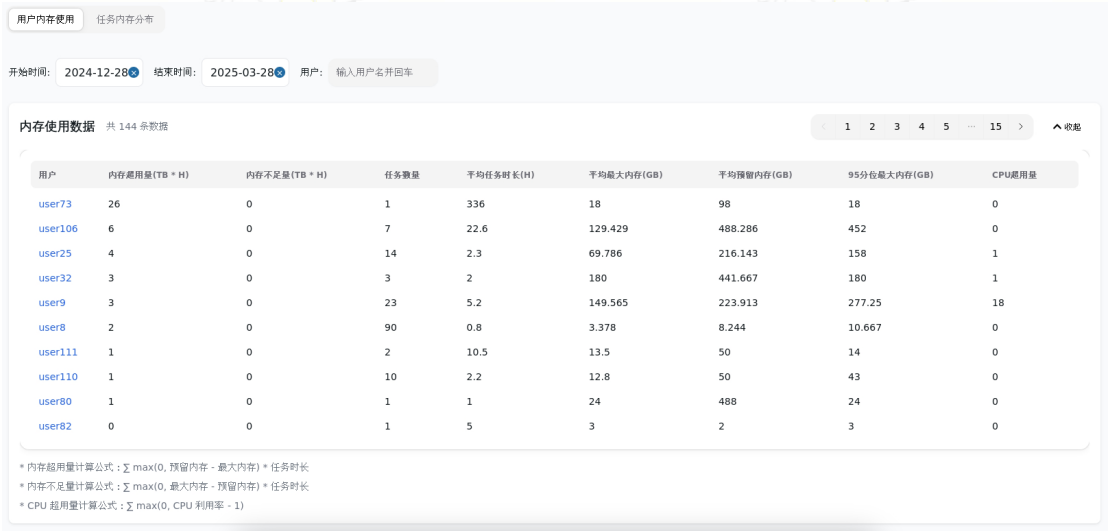
Web 提供两个 memory 分析界面， 一个是用户内存使用分析界面， 一个是任务内存分布分析界面。

4.6.1 用户内存使用

在浏览器中打开链接后， 可以直接看到用户内存使用界面。

表格默认展示用户最近三个月内提交的 job 的 memory 分析信息， 按照内存超用量（TB*H）进行倒序排序。可以根据需要重新设置表格上方的开始时间和结束时间， 或者在用户一栏输入具体的用户名对某个用户进行搜索。

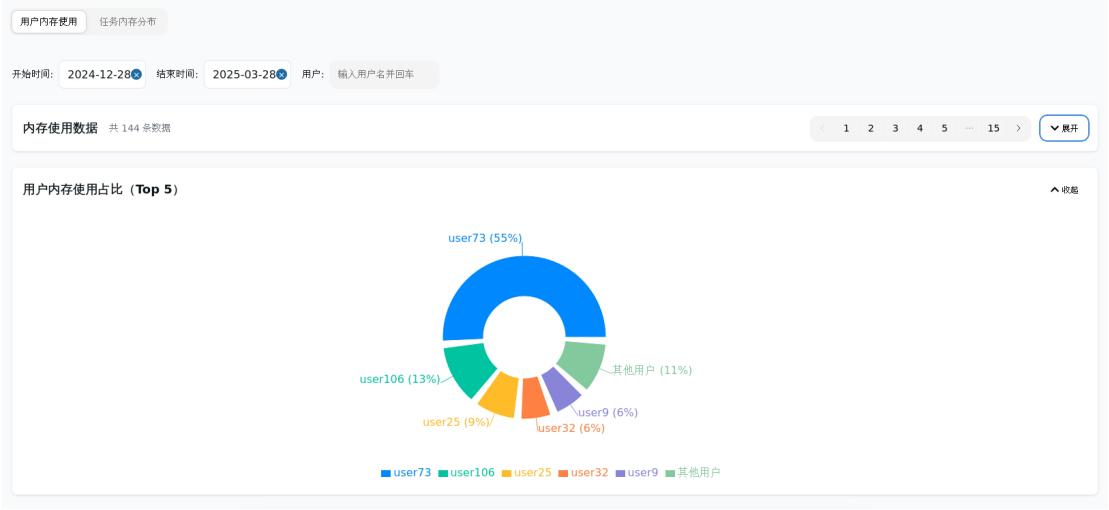
表格中的每一列都可以通过点击表格的标题栏， 使表格按该列顺序进行升序或者降序排序。



表格中的数据统计方法如下：

- 内存超用量：$\sum max(0, 预留内存 - 最大内存) * 任务时长$
- 内存不足量：$\sum max(0, 最大内存 - 预留内存) * 任务时长$
- 平均任务时长：$\frac{\sum 任务时长}{任务数量}$
- 平均最大内存：$\frac{\sum 最大内存}{任务数量}$
- 平均预留内存：$\frac{\sum 预留内存}{任务数量}$
- 95 分位最大内存：$\frac{\sum_{开始日期}^{结束日期} 某天\ 95\ 分位最大内存}{结束日期-开始日期}$
- CPU 超用量：$\sum max(0, CPU利用率 - 1)$

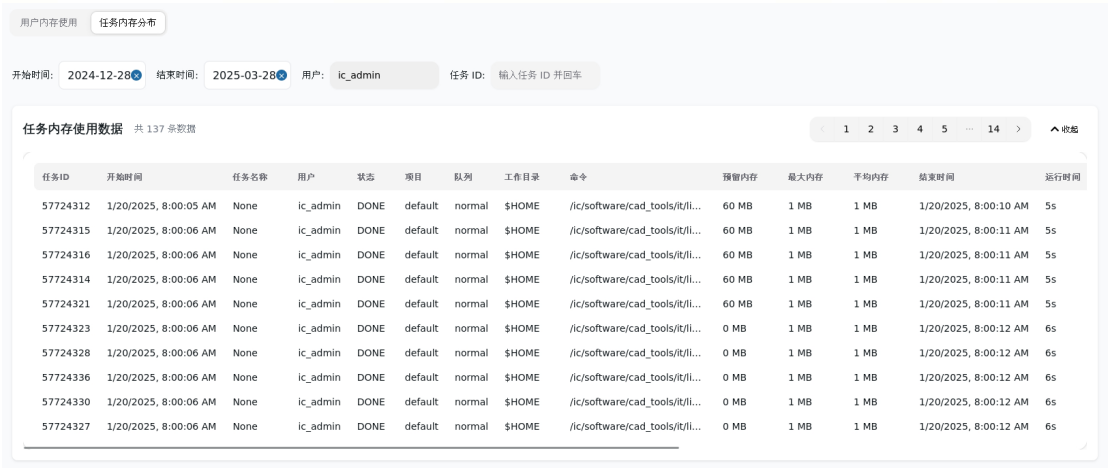
点击表格右上方的收起后，可以看到表格下方的饼状图。饼状图展示的是内存超用量最大的前五个用户， 和其他所有用户所占内存超用量总量的占比。



4.6.2 任务内存分布

点击用户内存分布表格中的任意一个用户， 界面将自动跳转到第二页，默认展示该用户近三个月的最近 1000 个 job 的详细信息。

开始时间和结束时间可以重新选择， 也可以在用户栏输入用户名， 查找对应用户的 job。任务 ID 栏填写 job id， 也可以查询对应 job 的详细信息。



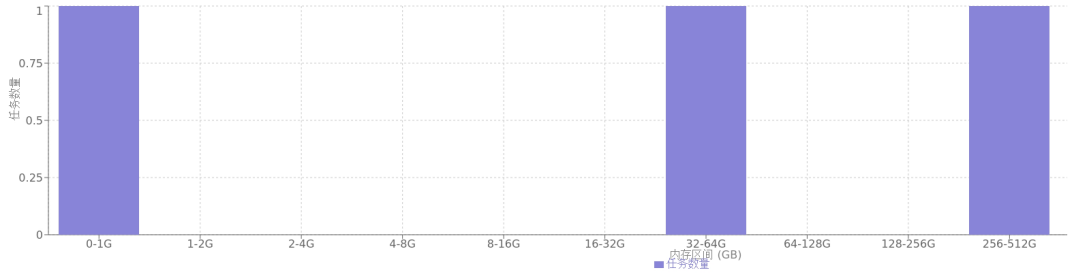
对于表格中展示的 job， 在表格的下方有对应的柱状图， 展示这些 job 使用的最大内存的在各个区间的分布。

用户内存使用 | 任务内存分布

开始时间: 2024-12-28 结束时间: 2025-03-28 用户: 任务 ID: 输入任务 ID 并回车

任务内存使用数据 共 4 条数据 1 展开

任务内存分布 收起



附录

附 1. 变更历史

日期	版本	变更描述	备注
2024.06	1.0	发布第一个版本。	
2024.08	1.0.1	支持从 lsfMonitor 的 job 采样目录中获取原始数据，取代 memPrediction 的采样行为。	
2025.03	1.1	支持 lsfMonitor 新的数据采样格式： sqlite 增加用户 memory 使用分析和 job memory 使用分析 Web 应用	

备注：小的 hotfix 不计入变更历史，bugfix 会实时 checkin 到 github 上。