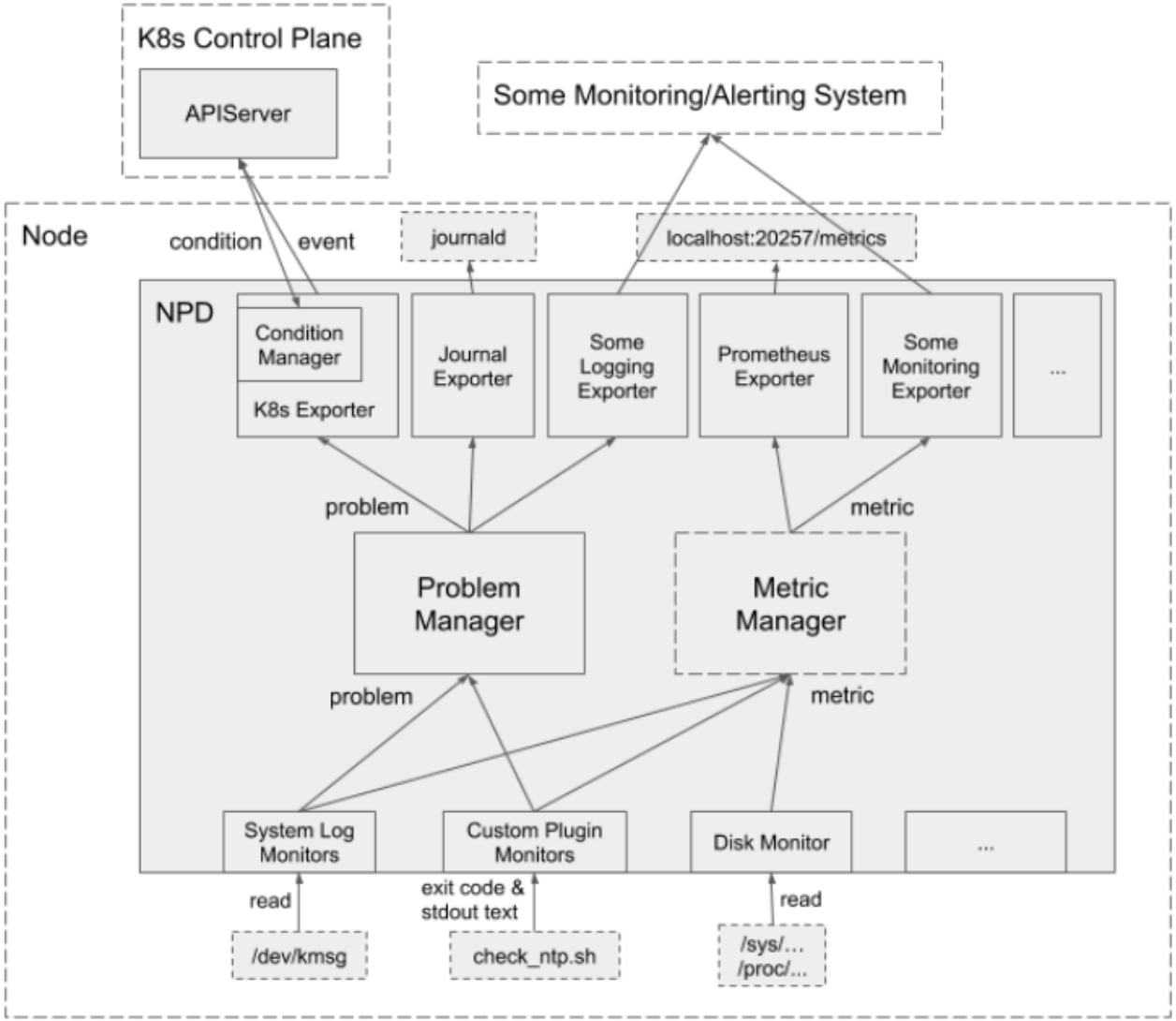
Predictive Maintenance

Topics

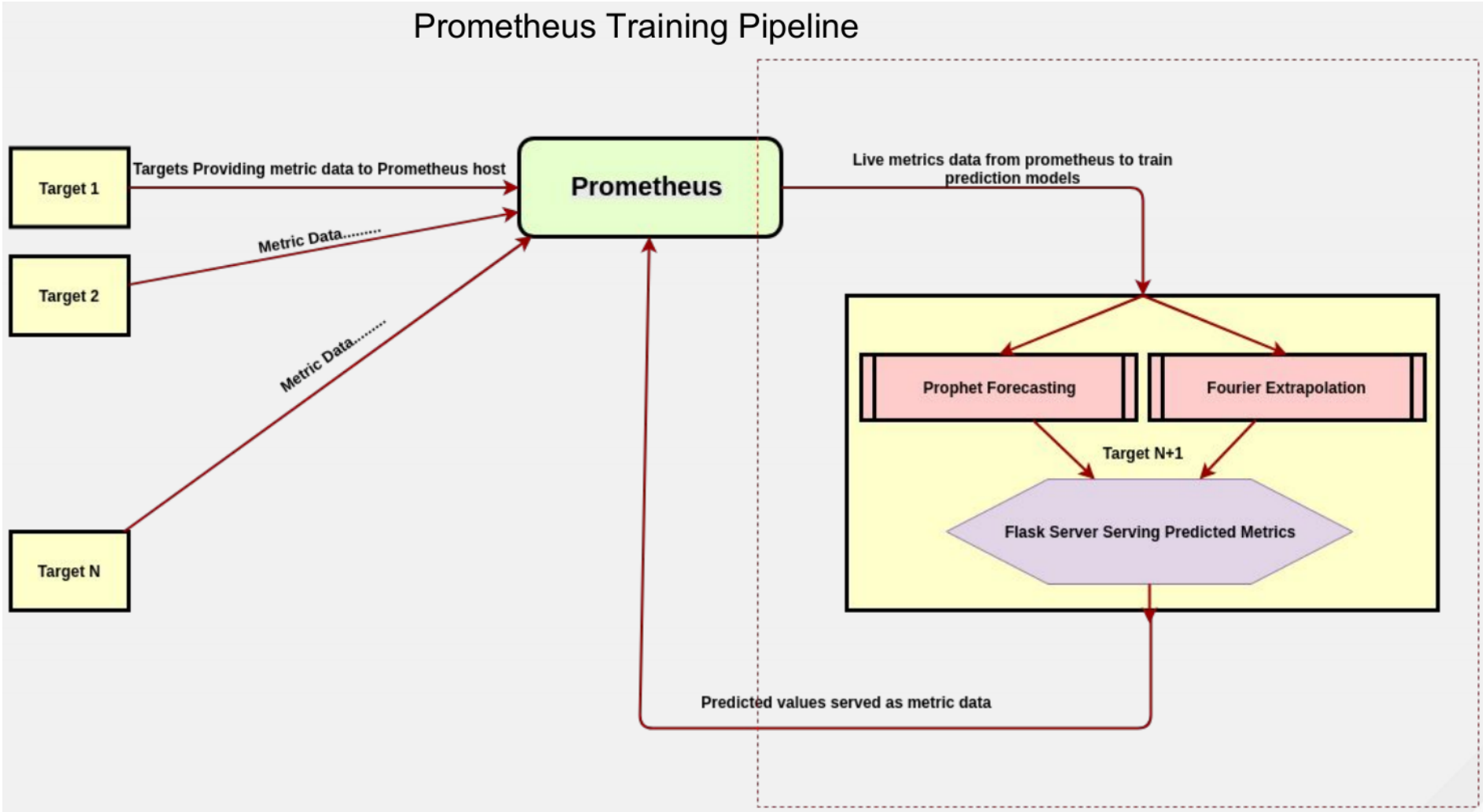
- Maintenance Scheduling
- Anomaly Detection
- Failure Prediction
- Improving Utilization
- Utilization Prediction
- AIOps platform
- CAP保证

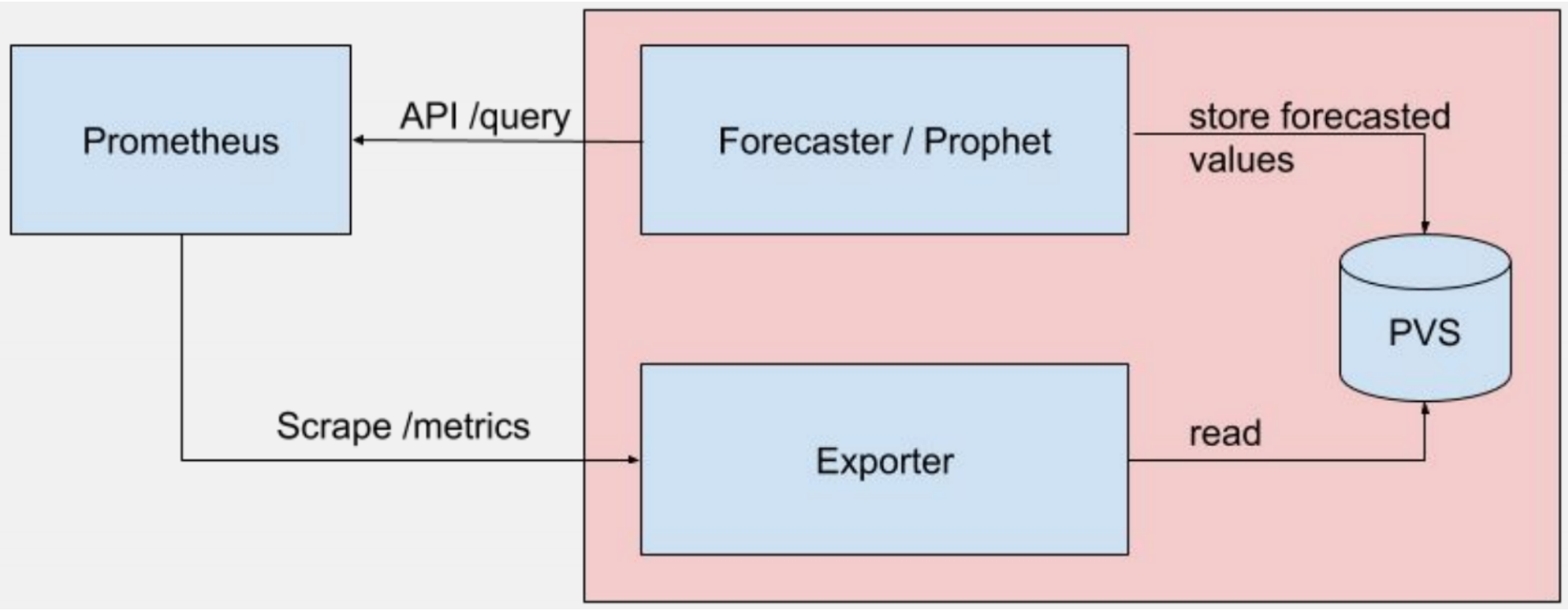
Python TensorFlow models serving with Go

- https://tech.travelaudience.com/training-tensorflow-models-in-python-and-serving-with-go-1b2a9386b0ff


K8s Control Plane
APIServer
Some Monitoring/Alerting System
Node
condition
event
journald
localhost:20257/metrics
NPD
Condition Manager
K8s Exporter
Journal Exporter
Some Logging Exporter
Prometheus Exporter
Some Monitoring Exporter
...
problem
metric
Problem Manager
Metric Manager
problem
metric
System Log Monitors
Custom Plugin Monitors
Disk Monitor
...
read
exit code & stdout text
read
/dev/kmsg
check_ntp.sh
/sys/…
/proc/...

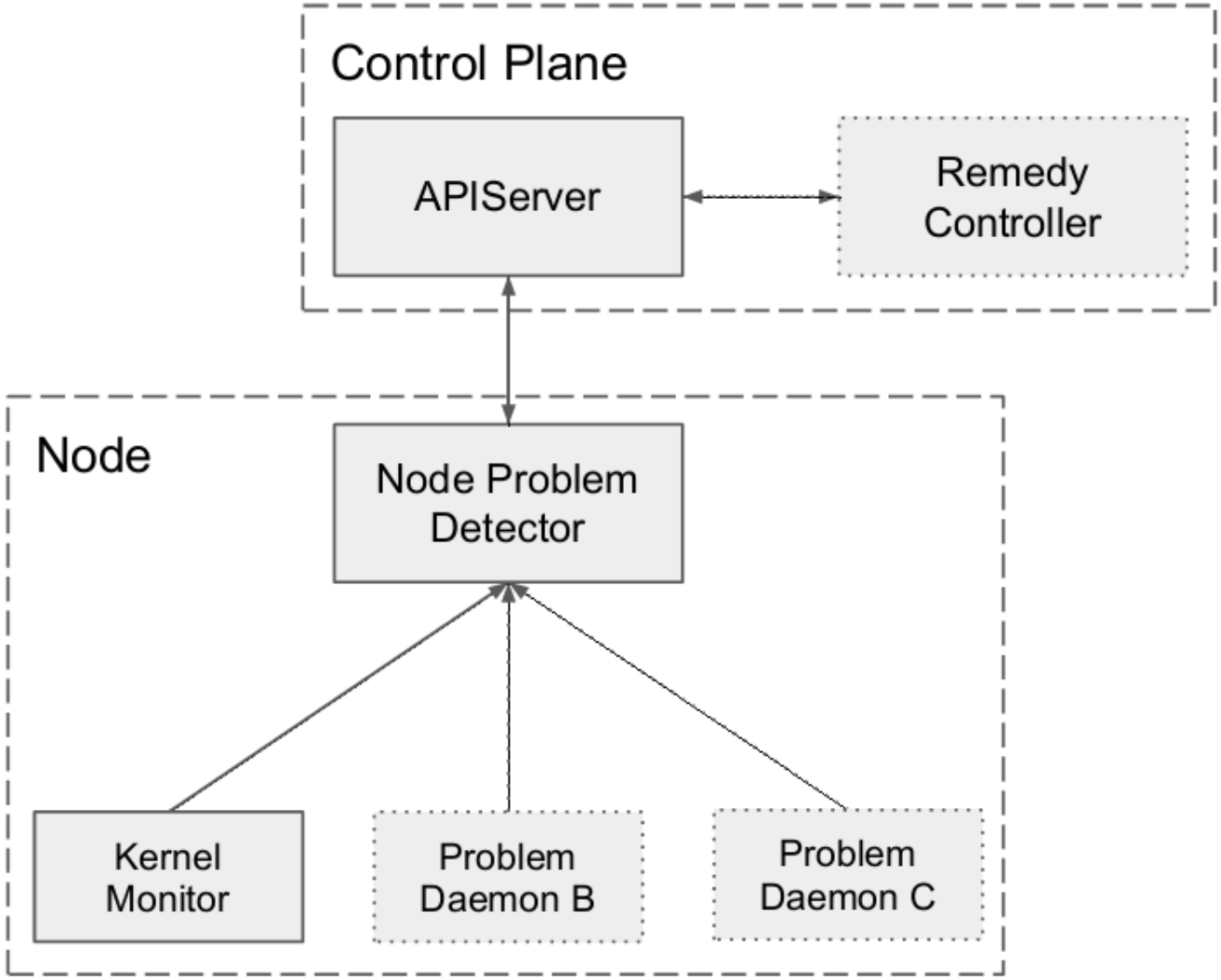
Training Pipeline



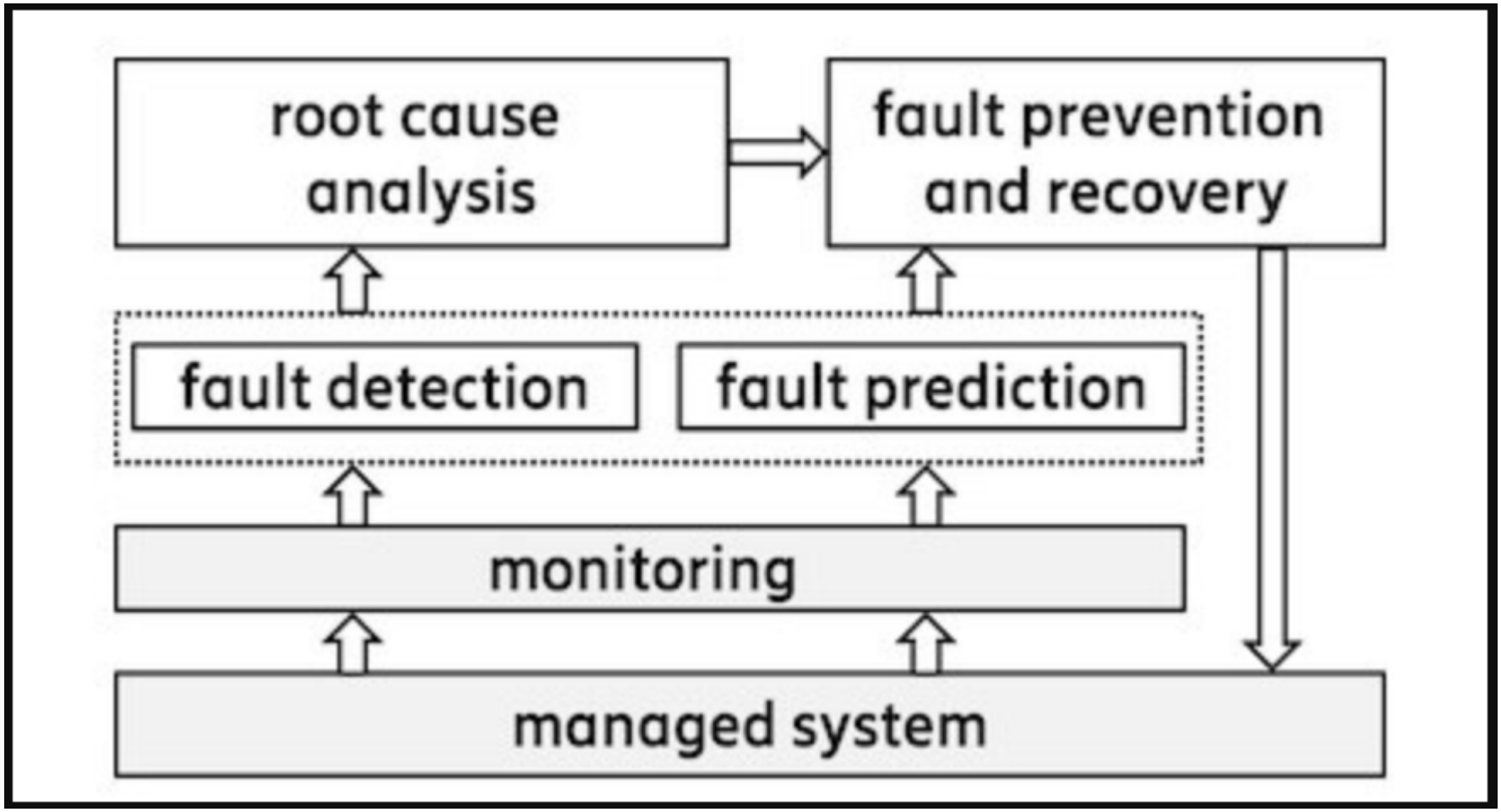

Prometheus
API /query
Forecaster / Prophet
store forecasted values
PVS
Scrape /metrics
Exporter
read

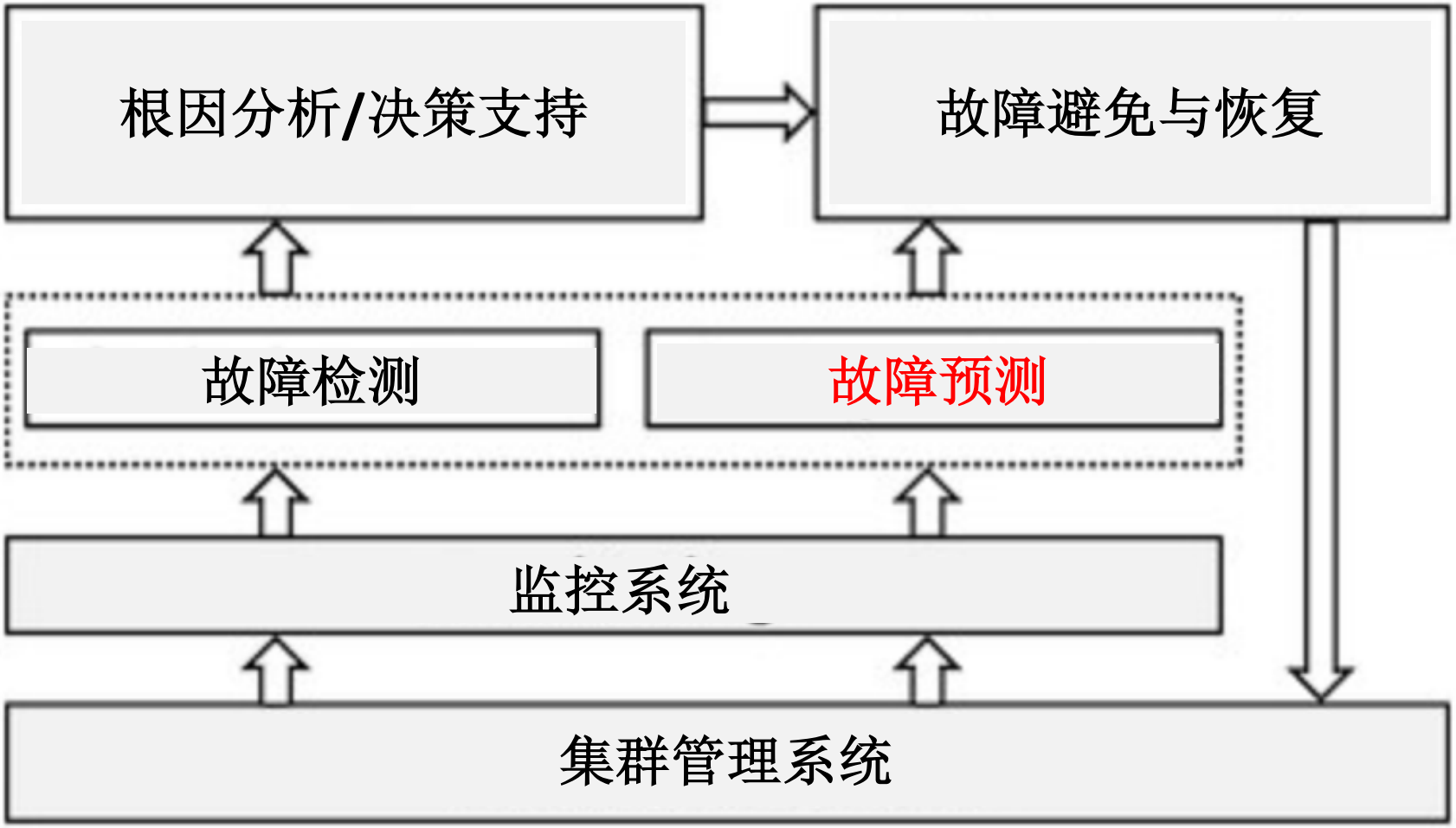
Remedy System

- Problem Manager
  - Problem Daemon
- Metric Manager


Control Plane
APIServer
Remedy Controller
Node
Node Problem Detector
Kernel Monitor
Problem Daemon B
Problem Daemon C

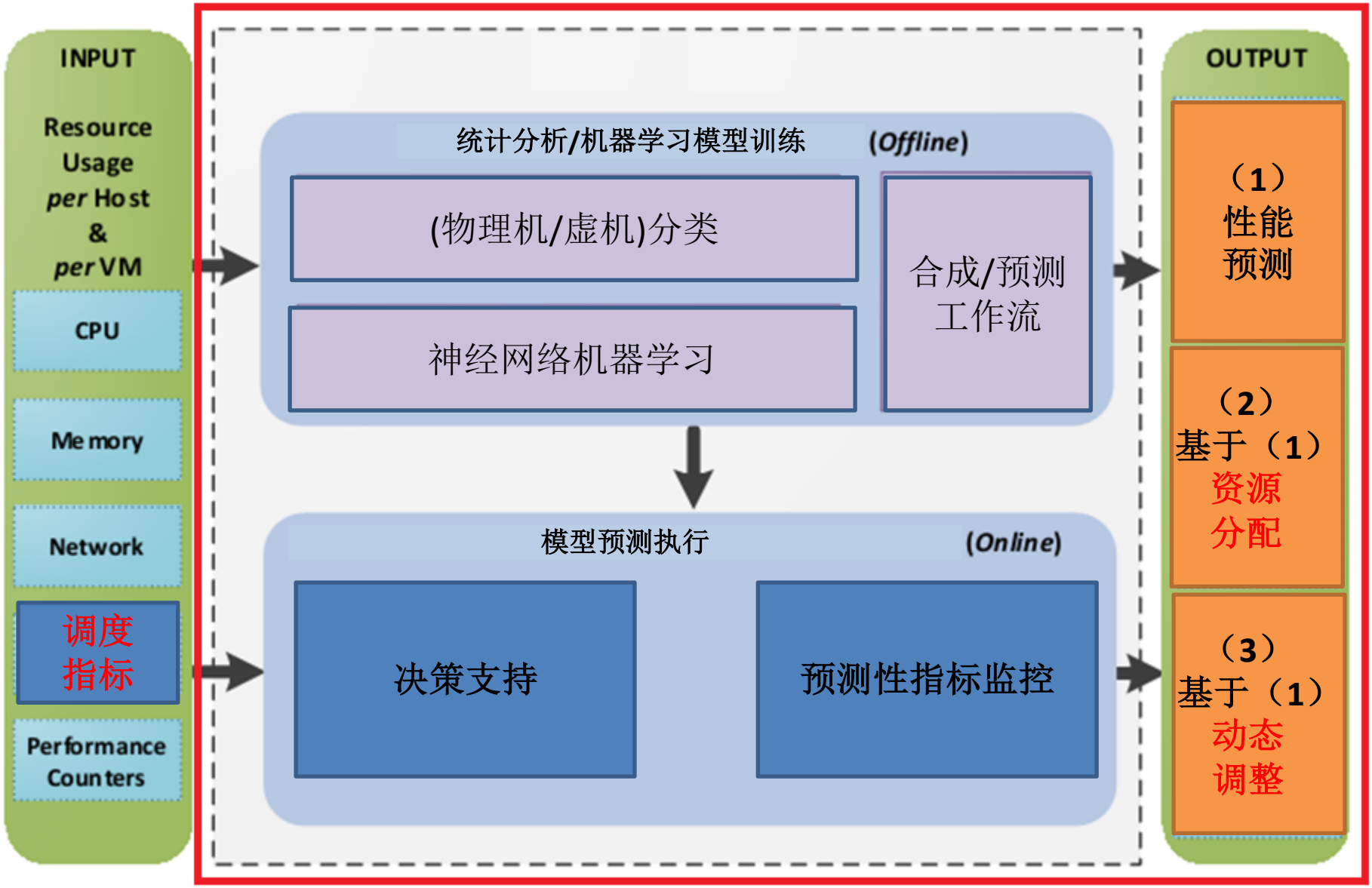
Fault Management System

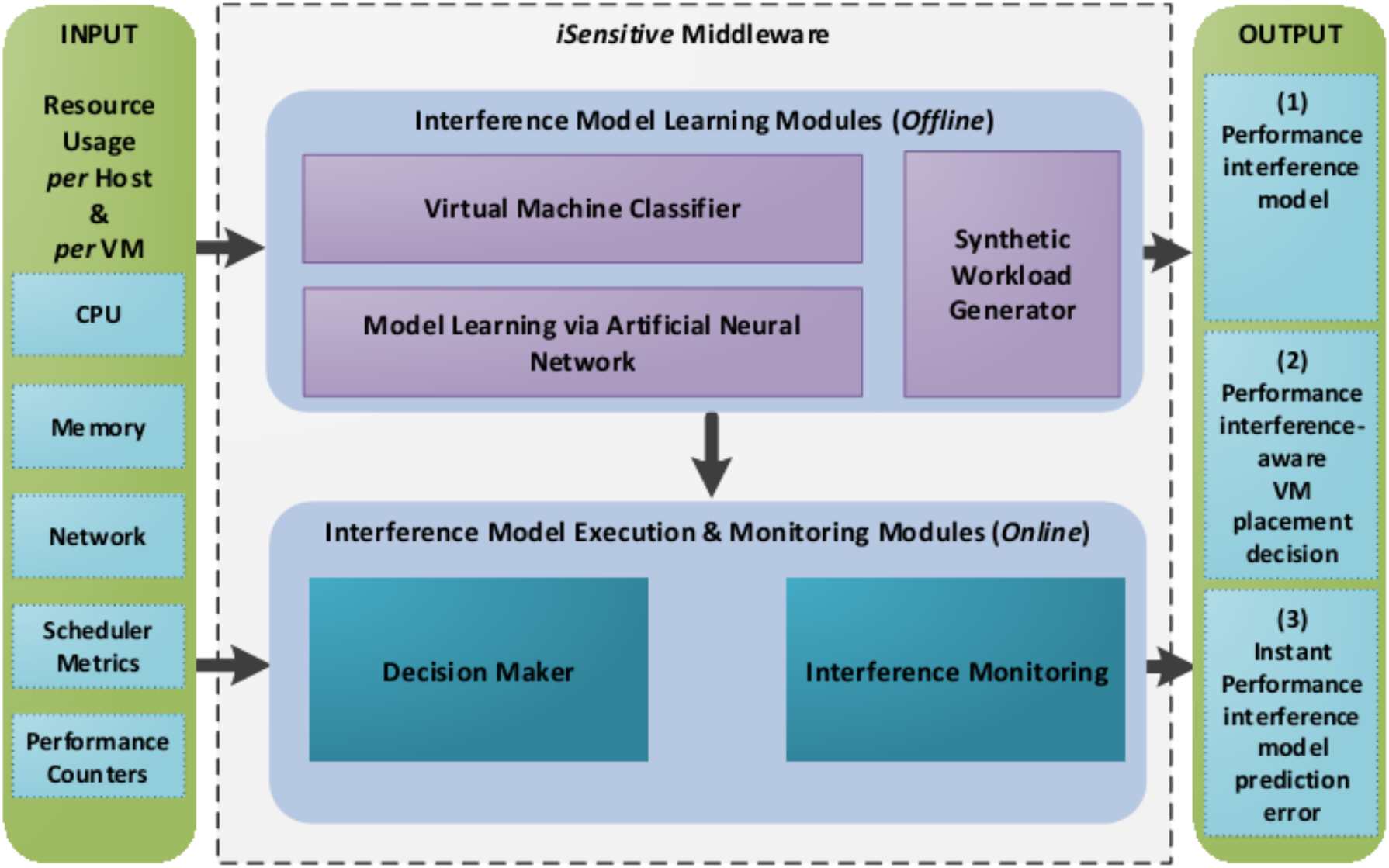


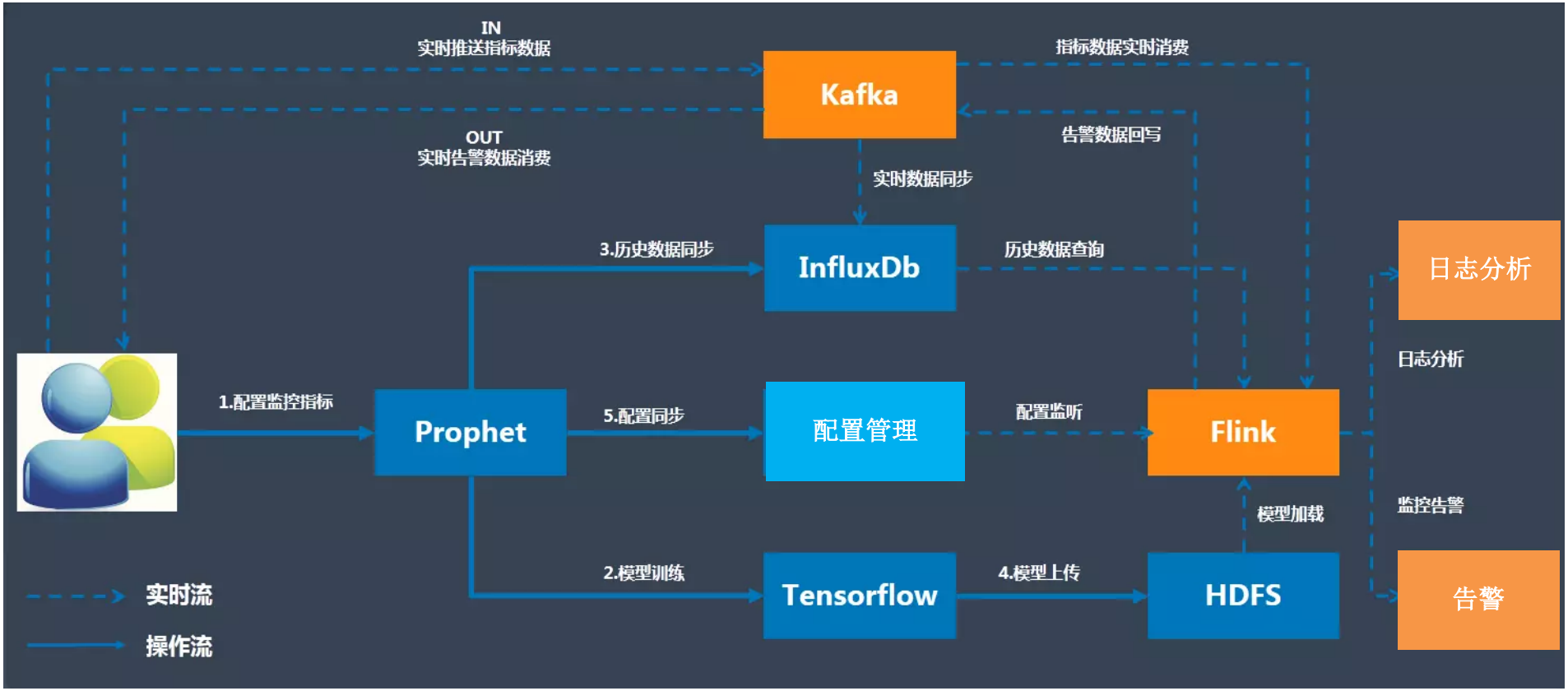

根因分析/决策支持
故障避免与恢复
故障检测
故障预测
监控系统
集群管理系统

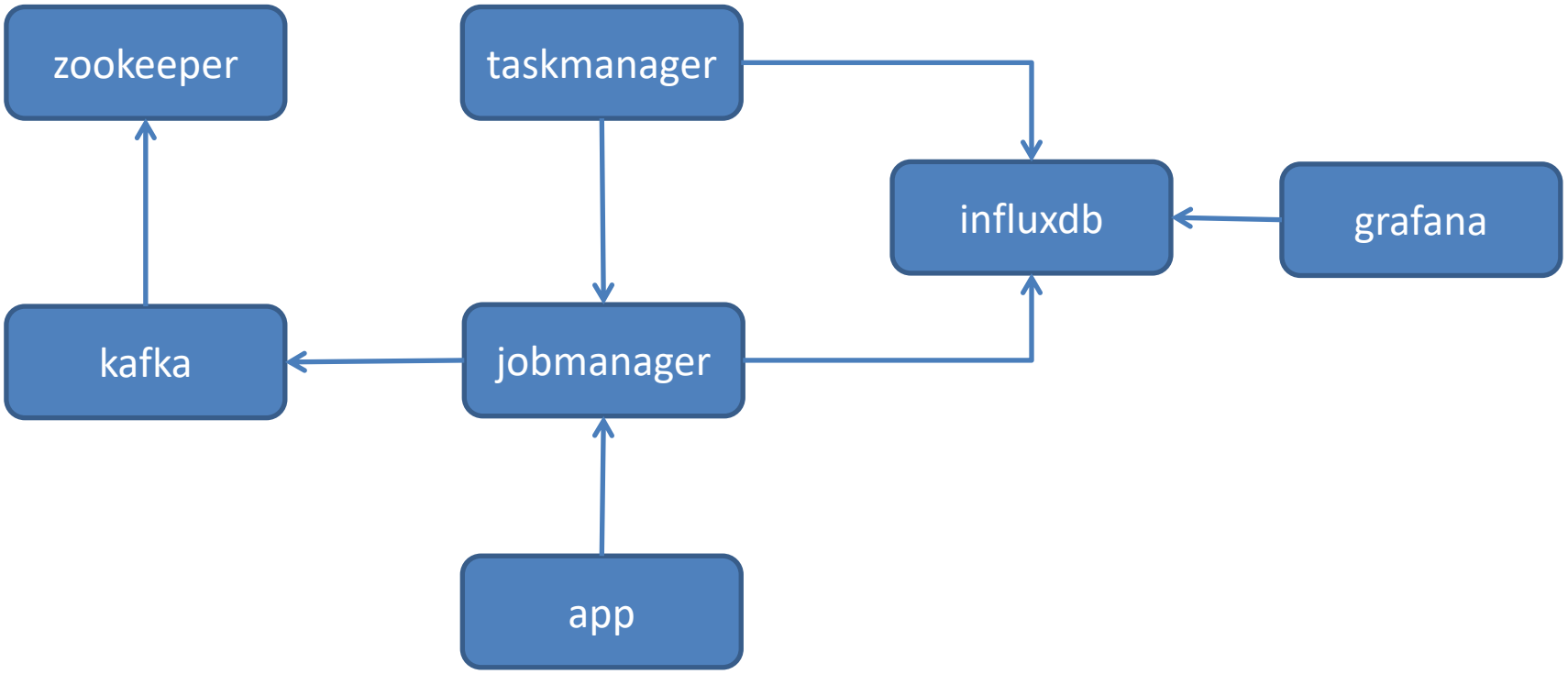
Resource Usage Prediction

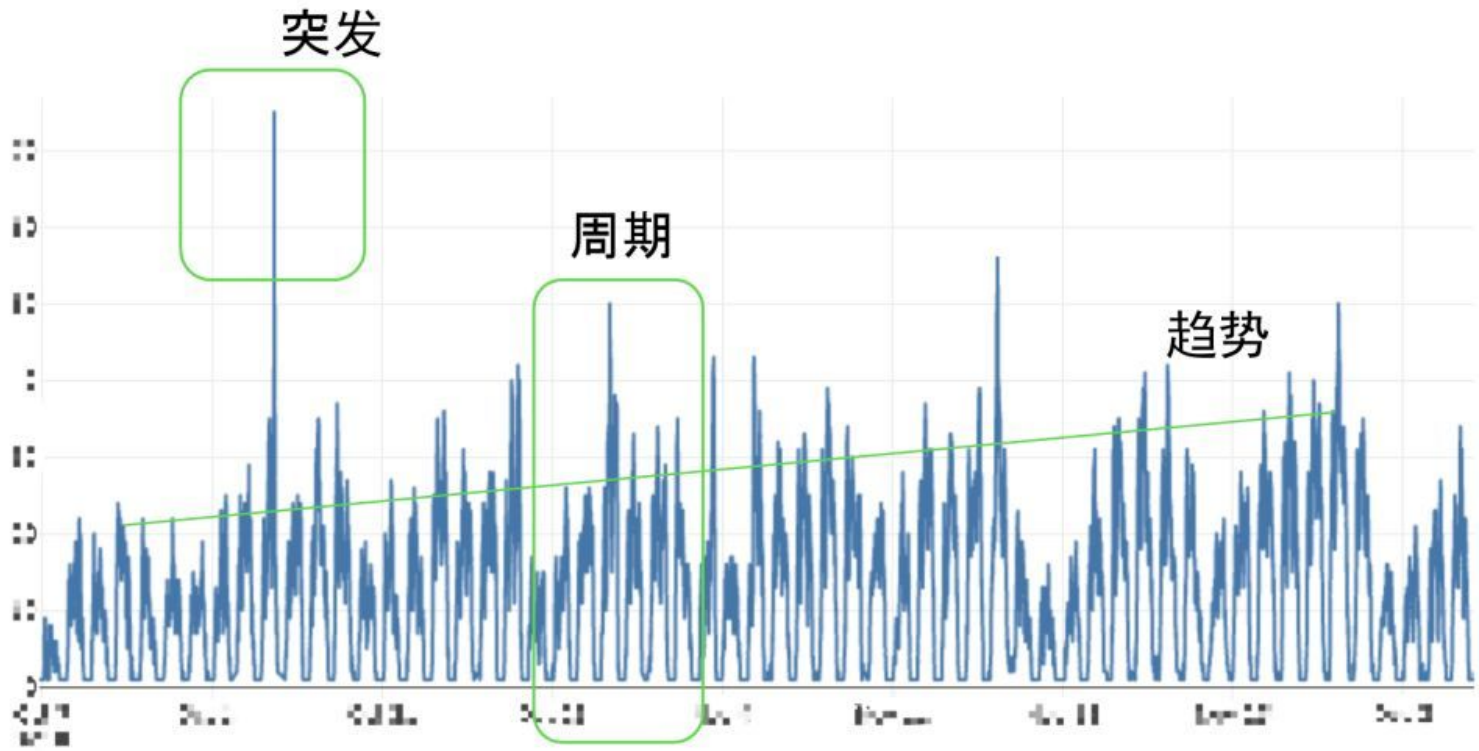
- Two layers, feed forward ANN
  - Cpu usage ratio, Memory usage ratio
  - Run vms
  - Ultility model

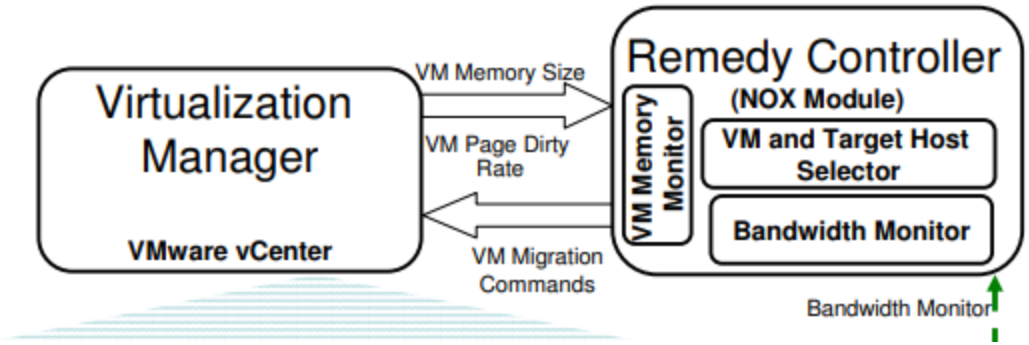

INPUT
Resource Usage per Host & per VM
CPU
Memory
Network
调度指标
Performance Counters
统计分析/机器学习模型训练
(Offline)
(物理机/虚机)分类
神经网络机器学习
合成/预测工作流
模型预测执行
(Online)
决策支持
预测性指标监控
OUTPUT
（1）性能预测
（2）基于（1）资源分配
（3）基于（1）动态调整

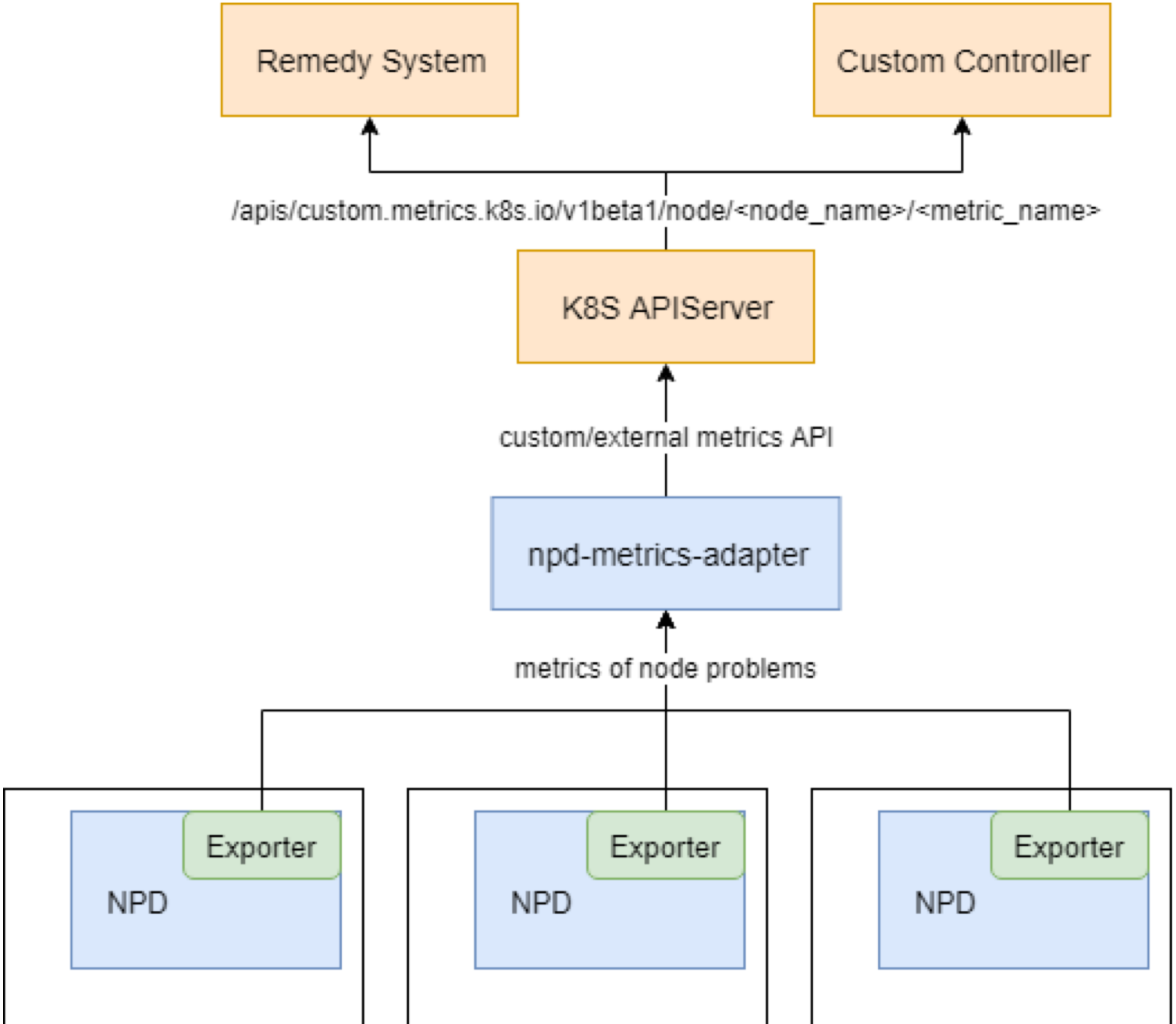

INPUT
Resource Usage per Host & per VM
CPU
Memory
Network
Scheduler Metrics
Performance Counters
iSensitive Middleware
Interference Model Learning Modules (Offline)
Virtual Machine Classifier
Model Learning via Artificial Neural Network
Synthetic Workload Generator
Interference Model Execution & Monitoring Modules (Online)
Decision Maker
Interference Monitoring
OUTPUT
(1) Performance interference model
(2) Performance interference-aware VM placement decision
(3) Instant Performance interference model prediction error


IN
实时推送指标数据
Kafka
指标数据实时消费
OUT
实时告警数据消费
告警数据回写
实时数据同步
3.历史数据同步
InfluxDb
历史数据查询
日志分析
日志分析
1.配置监控指标
Prophet
5.配置同步
配置管理
配置监听
Flink
模型加载
监控告警
2.模型训练
Tensorflow
4.模型上传
HDFS
告警
实时流
操作流


zookeeper
taskmanager
influxdb
grafana
kafka
jobmanager
app


突发
周期
趋势


Virtualization Manager
VMware vCenter
VM Memory Size
VM Page Dirty Rate
VM Migration Commands
Remedy Controller
(NOX Module)
VM Memory Monitor
VM and Target Host Selector
Bandwidth Monitor
Bandwidth Monitor


Remedy System
Custom Controller
/apis/custom.metrics.k8s.io/v1beta1/node/<node_name>/<metric_name>
K8S APIServer
custom/external metrics API
npd-metrics-adapter
metrics of node problems
Exporter
NPD
Exporter
NPD
Exporter
NPD

Task分析

Case	Py	Input	Output
task_uasge_500_preprocess	data_preprocess.py	task_usage_500_title.csv	test.csv
	data_inverse.py	test_data.csv	test_data_inversed_bycode.csv
	data_GRU_run.py	dataset_inversed.csv	plot
	google_cluster_dataset_500_predict/ google_cluster_dataset_500_predict.py	test_data_inversed.csv	plot
task_usage_total_preprocess	preprocess/with_maid/ data_predict.py	task_usage_preprocessed.csv	plot
	preprocess/with_maid/ data_preprocessing.py	task_usage.csv	task_usage_preprocessed_ud.csv
	preprocess/with_maid/ dataset_drop_mean.py	task_usage_preprocessed_ud.csv	task_usage_preprocessed.csv
	preprocess/Without_maid(ignore)/ data_total_preprocessing.py	task_usage.csv	task_usage_total_inversed_bycode.csv
	optimizer/Bidirectional_Swish/ bidirectional_swish.py	task_usage_preprocessed.csv	plot
	optimizer/Swish_activate/ swish_activate.py	task_usage_preprocessed.csv	plot

Trace分析

Case	Py	Input	Output
cluster	Alibaba_cluster/ Alibaba_cluster_LSTM_predict.ipynb	Machine_usage_groupby.csv	Plot(failed)
	Alibaba_cluster/ Alibaba_cluster_GRU_predict.ipynb	Machine_usage_groupby.csv	plot
	Alibaba_cluster/ Alibaba_cluster_drop_predict.ipynb	Machine_usage_groupby.csv	plot
cluster_predict	Alibaba_cluster_predict/code/ preprocess.py	machine_usage_50000.csv	Machine_usage_fill.csv
	Alibaba_cluster_predict/code/ Averange_time_process.py	Machine_usage_fill.csv	Machine_usage_groupby.csv
	Alibaba_cluster_predict/code/ data_run.py	Machine_usage_groupby.csv	plot & RMSE
	Alibaba_cluster_predict/code/ Plot_cluster_data.py	Machine_usage_groupby.csv	plot
cluster_predict_Total	cluster_predict_Total/ Alibaba_cluster_predict_runafterprocession.py	Machine_usage_groupby.csv	
	cluster_predict_Total/ data_give_head.py	machine_usage.csv	machine_usage_headed.csv
	cluster_predict_Total/ data_test_head.py	Machine_usage_groupby_100.csv	nil
	cluster_predict_Total/ data_preprocess.py	machine_usage_headed.csv	Machine_usage_groupby.csv

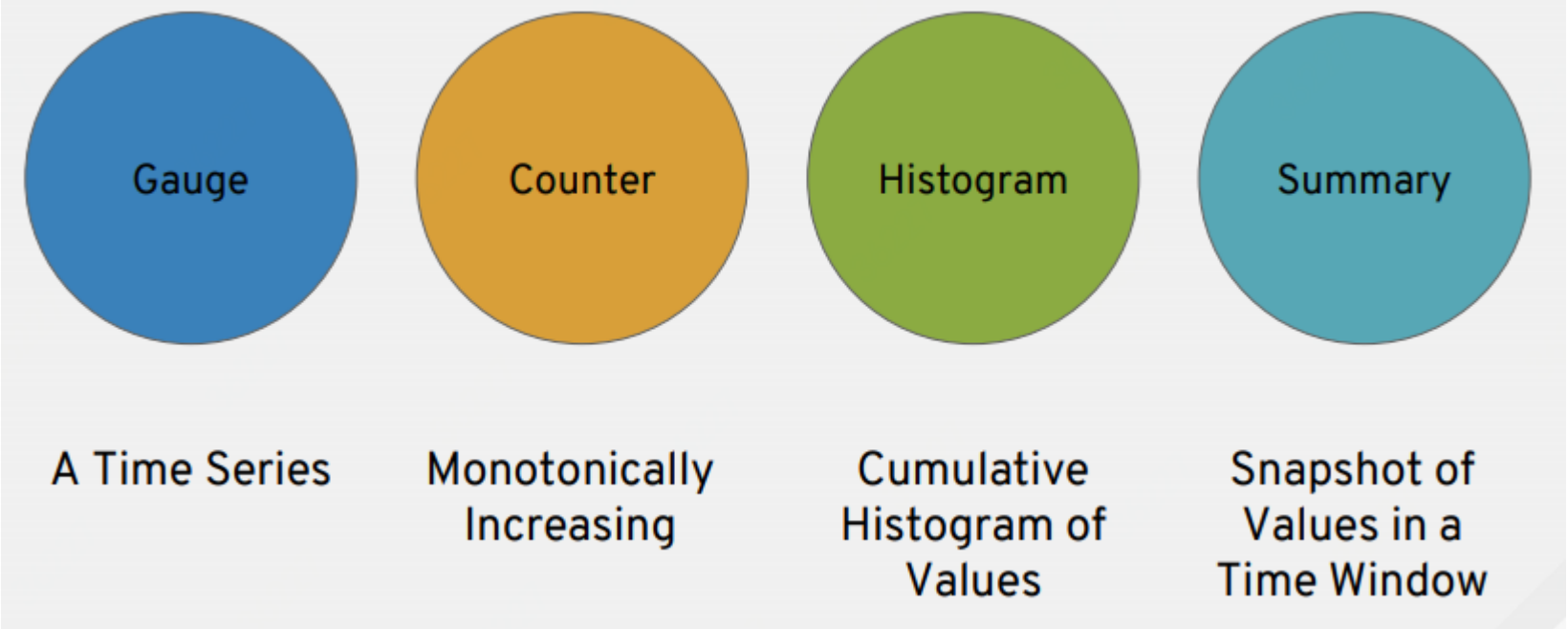
Trace分析

Case	Py	Input	Output
cluster_predict_Total	cluster_predict_Total/ prediction_model.py	Machine_usage_groupby.csv	plot & RMSE & MSE
	cluster_predict_Total/ test_mse_predict.py	Machine_usage_groupby.csv	plot & RMSE & MSE (error)
cluster_predict_compare	Train_40000/COV+GRU+GRU+D+D+DROP+E50/ COV+GRU+GRU+D+D+DROP+E50.py	Machine_usage_groupby.csv	Plot & RMSE &MSE
	Train_40000/Bidirectional_Swish/ bidirectional_swish.py	Machine_usage_groupby.csv	Plot & RMSE &MSE
	Train_20000/COV+GRU+GRU+D+D+E50/ COV+GRU+GRU+D+D+E50.py	Machine_usage_groupby.csv	Plot & RMSE &MSE
	Train_20000/COV+LSTM+LSTM+D+D+DROP+E50/ alibaba_lstm_predict.py	Machine_usage_groupby.csv	Plot & RMSE &MSE
	Train_20000/GRU+D+D+E50/ GRU+D+D+E50.py	Machine_usage_groupby.csv	Plot & RMSE &MSE
	Train_20000/GRU+D+E50/ GRU+D+E50.py	Machine_usage_groupby.csv	Plot & RMSE &MSE
	Train_20000/GRU+GRU+D+D+DROP+E50/ GRU+GRU+D+D+DROP+E50.py	Machine_usage_groupby.csv	Plot & RMSE &MSE
	Train_20000/GRU+GRU+D+D+E50/ GRU+GRU+D+D+E50.py	Machine_usage_groupby.csv	Plot & RMSE &MSE

Remedy-controller

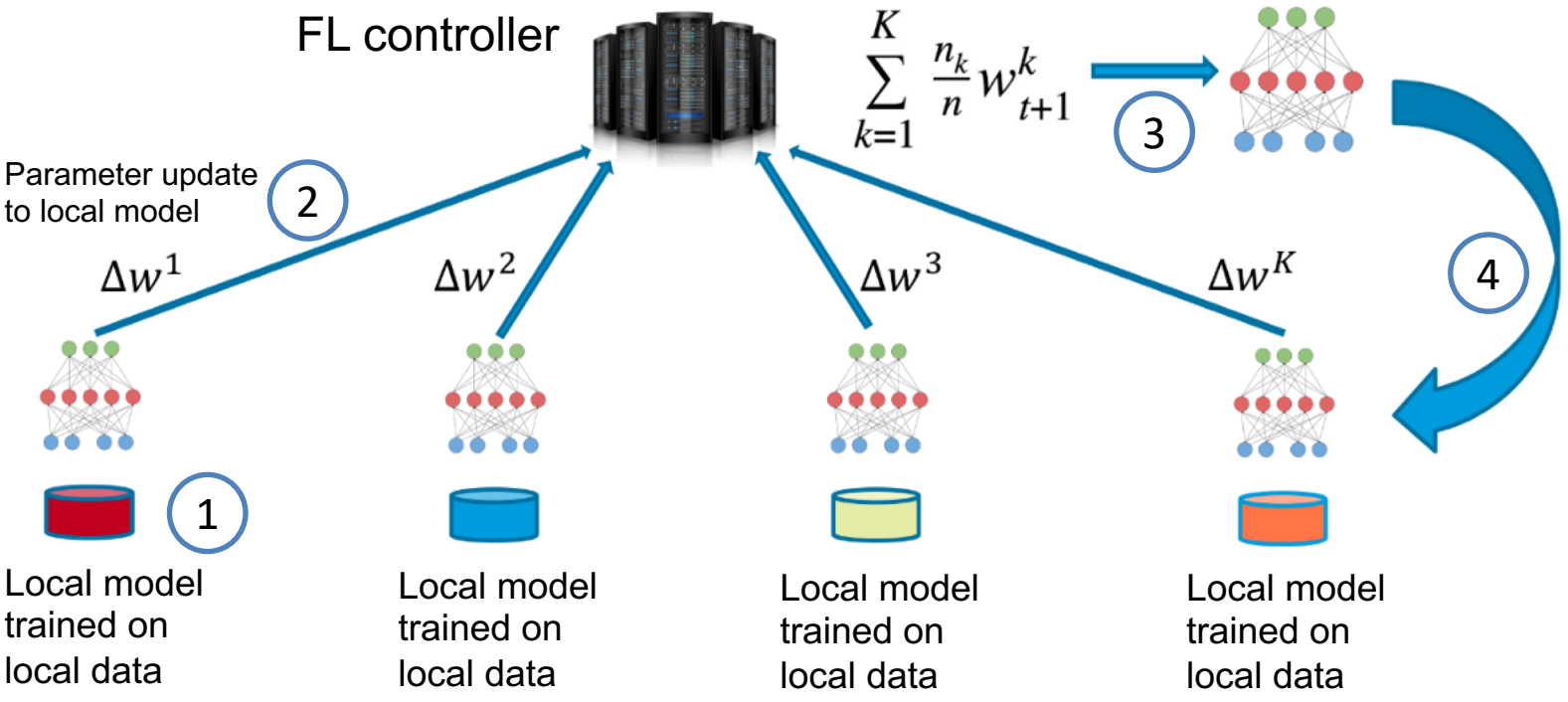
- [draino]
- [descheduler]
- [debug-application-cluster]

Prometheus Metric Types



- Anomaly detector
  - https://github.com/AICoE/prometheus-anomaly-detector

Federated Learning

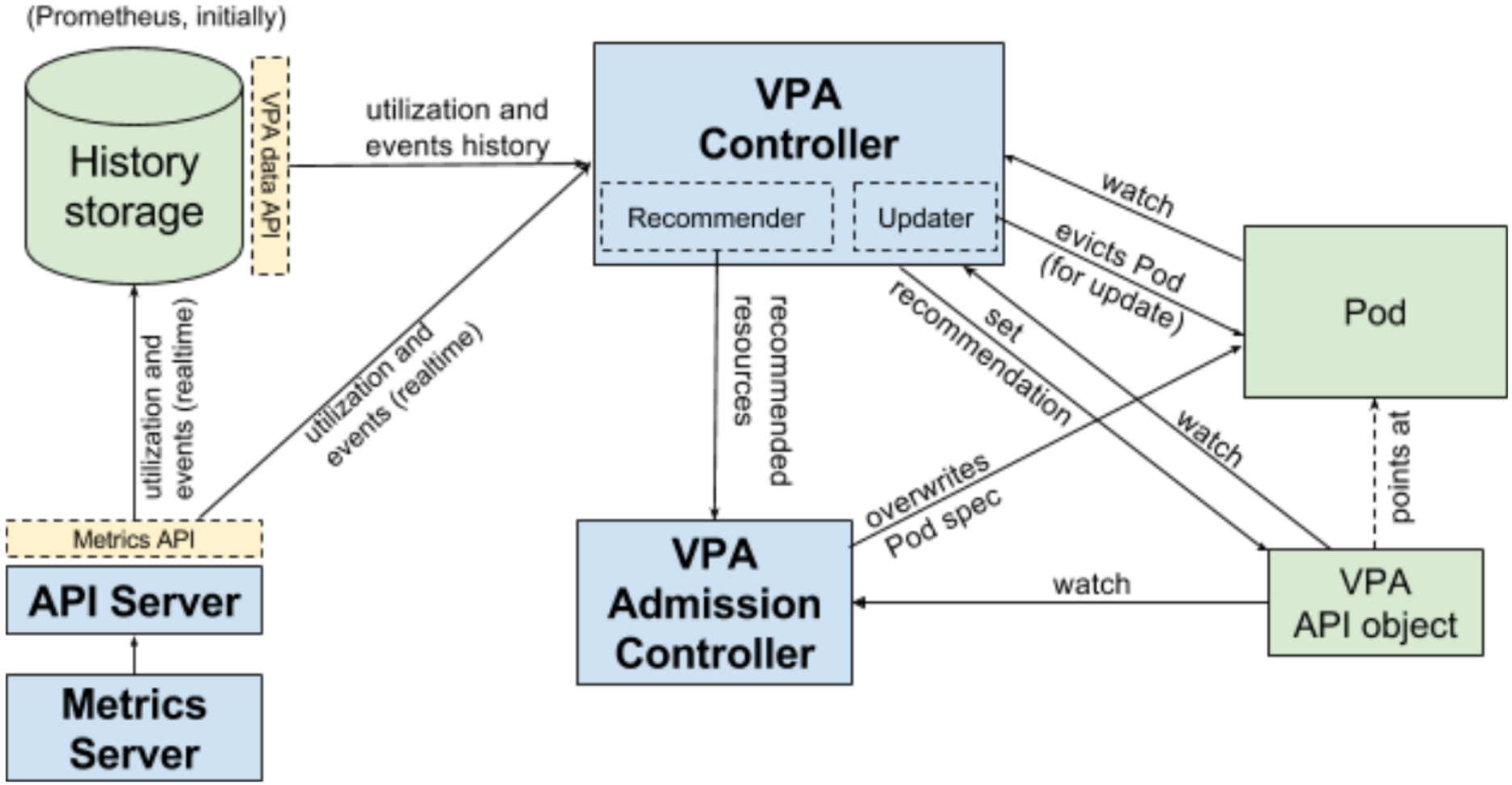


Source: https://miro.medium.com/max/1400/1*HaH611vAy2eB1e42vz3X4g.png

最优建模

- COV+GRU+GRU+D+D++DROP+E50

VPA(Autopilot)



ML platform

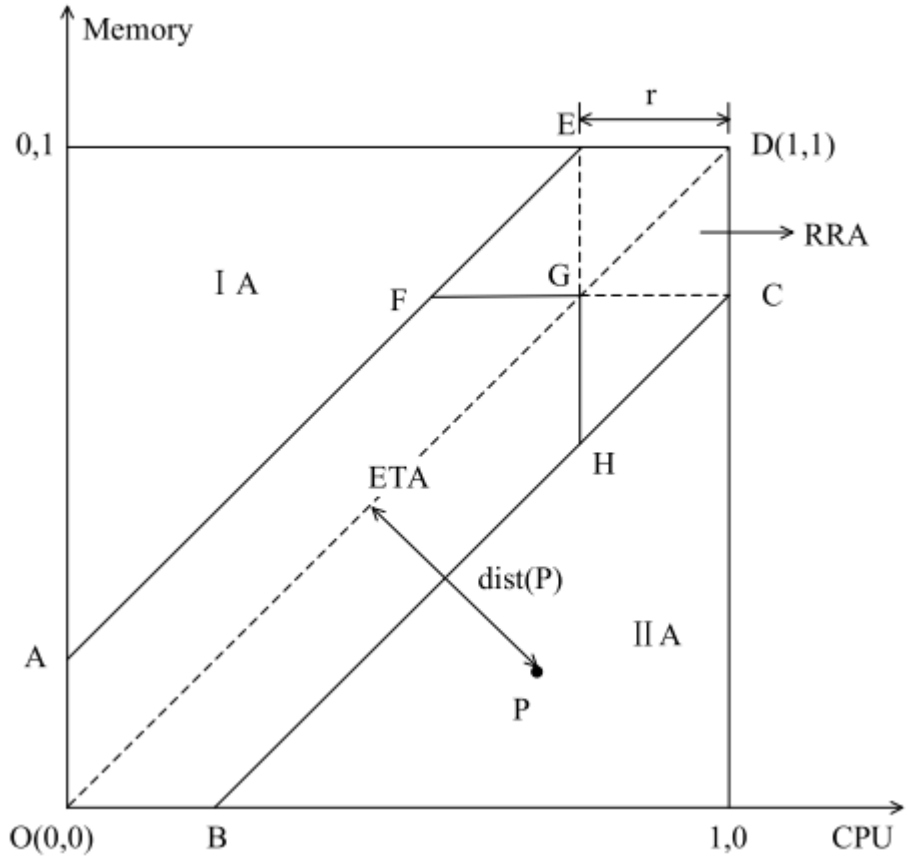
- maro on k8s

Defragmenting the Cloud

- 资源碎片化建模
  - 有CPU无MEM
  - 无CPU有MEM

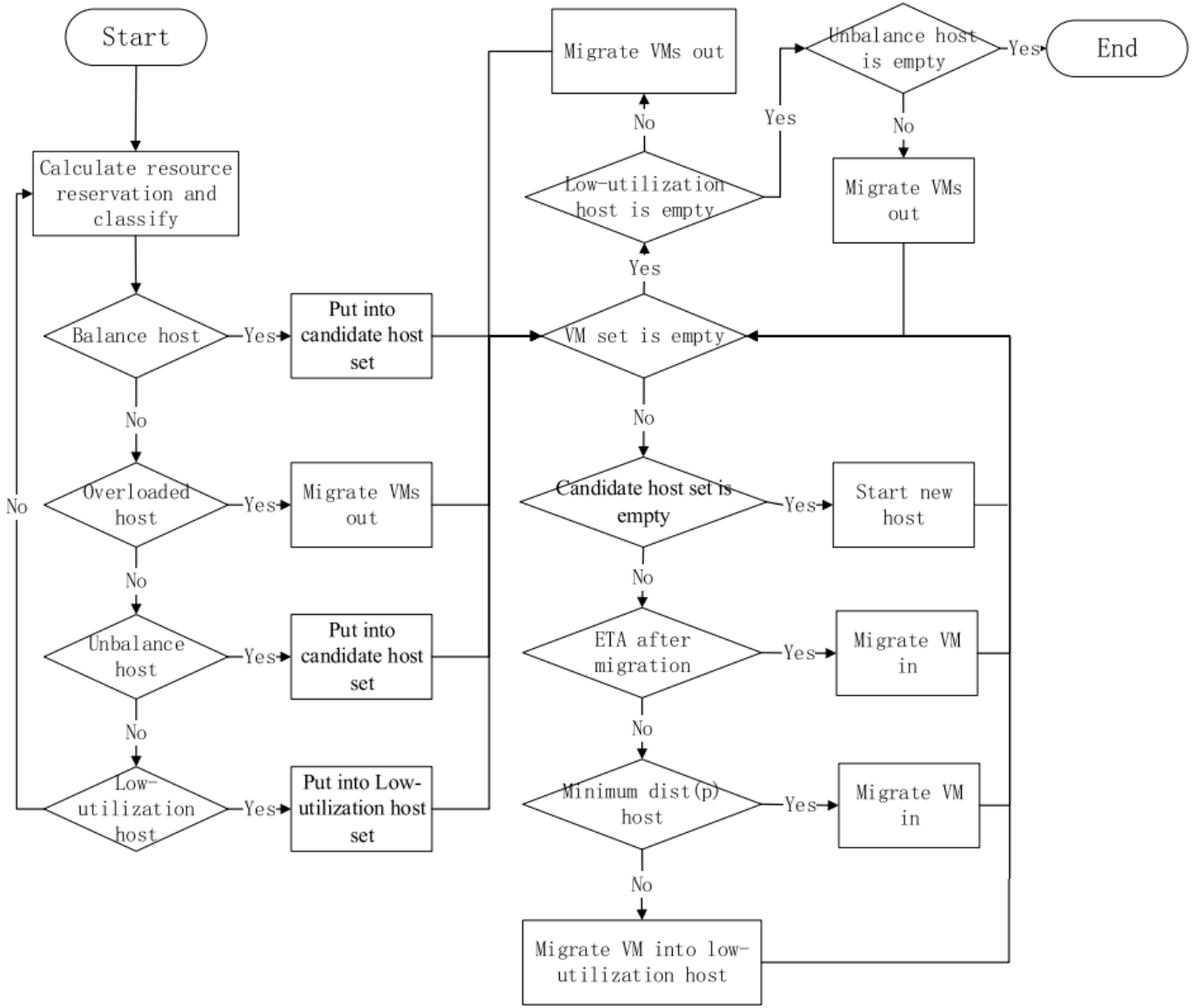
利用率描述

- RRF index = minx{(空白)/（TOTAL）}



$$f(x,y) = \begin{cases} RRA, & (1-x) \le r \& (1-y) \le r \\ ETA, & dist(p) \le \dfrac{\sqrt{2}}{2} r \\ & \&(1-x) > r \& (1-y) > r \\ IA\ or\ IIA, & otherwise \end{cases}$$

动态迁移流程


Start
Calculate resource reservation and classify
Balance host
Yes
Put into candidate host set
No
Overloaded host
Yes
Migrate VMs out
No
Unbalance host
Yes
Put into candidate host set
No
Low-utilization host
Yes
Put into Low-utilization host set
No
Migrate VMs out
No
Low-utilization host is empty
Yes
Yes
Unbalance host is empty
Yes
End
No
Migrate VMs out
Yes
VM set is empty
No
Candidate host set is empty
Yes
Start new host
No
ETA after migration
Yes
Migrate VM in
No
Minimum dist(p) host
Yes
Migrate VM in
No
Migrate VM into low-utilization host

rms all	Not.ready	Ready	optimes
23628	54*	23574	11-16

jcs独部	jks-jcs混部	jks独部	baremetal	网络不通	操作合计	RMS节点总数
8087	**15402**	**4**	**78**	**3**	**23574**	23574

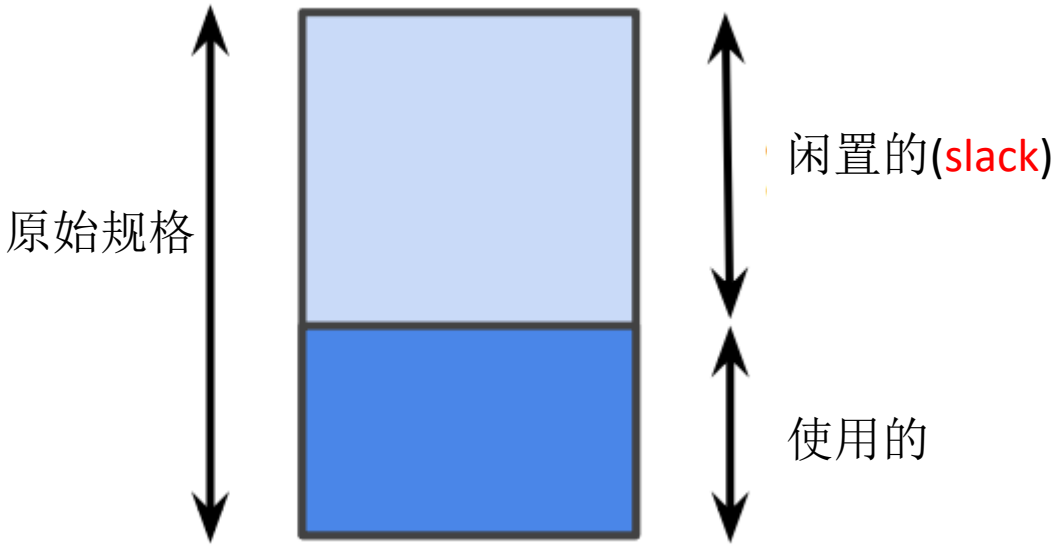
*

SUCC,10.217.174.172
SUCC,10.217.35.215
SUCC, 10.217.94.186

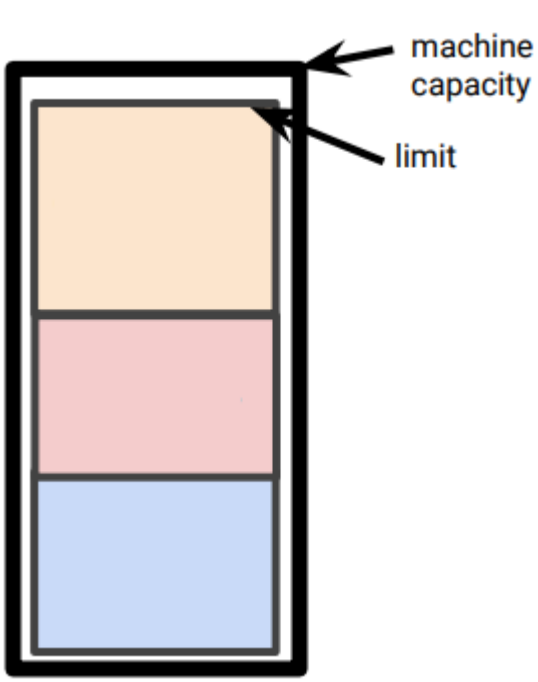
Limit预测

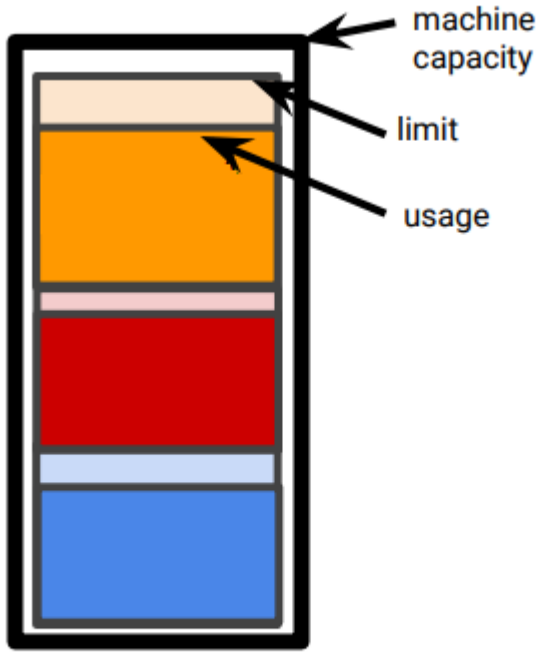
- 原理
- 场景
- 实验
- 测评

问题背景



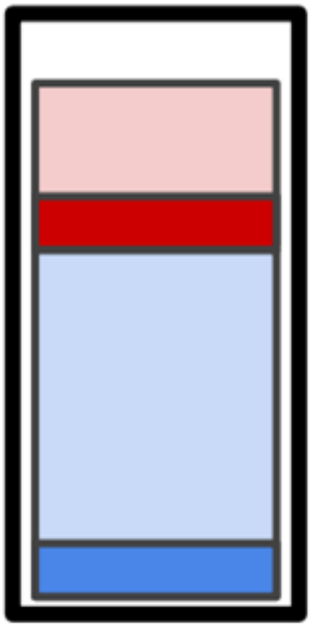
分析





precise
limits

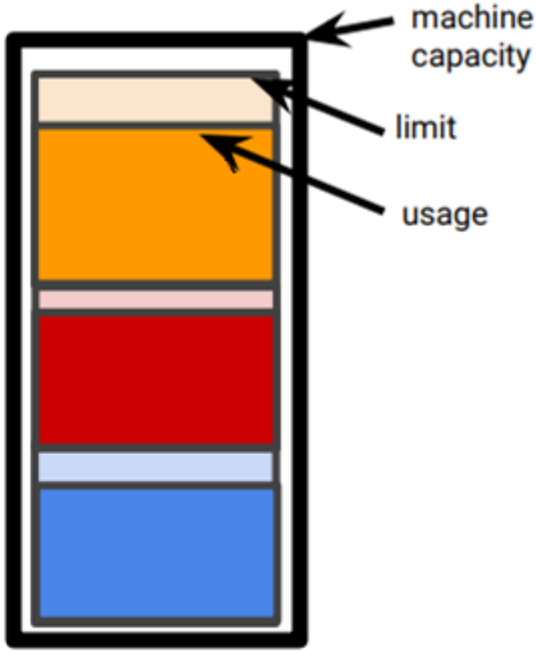
面对场景



resource are wasted (underallocated machine)

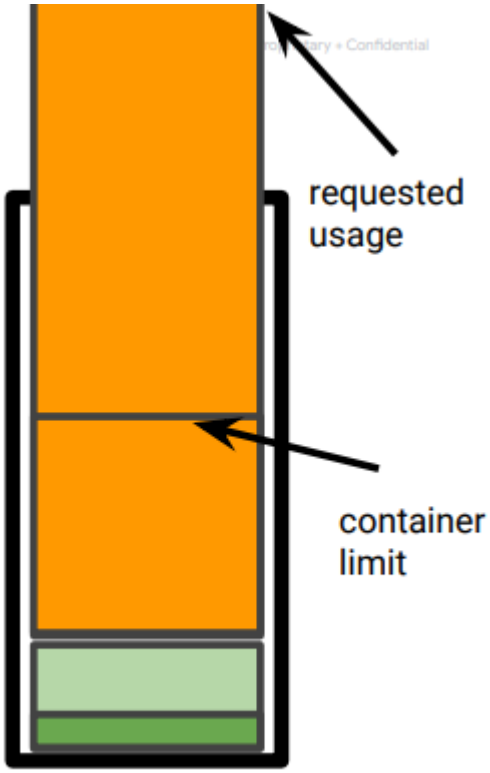
bad!

资源浪费



precise limits

good!



还是bad! 资源超了

实际要点

- 如何准确的预测资源水位？
- 一旦预测不准确，就会：
  - 影响 Qos，CPU可能还好
  - 内存如果预测不准，会触发OOM
    - 这种行为不可接受

解决办法

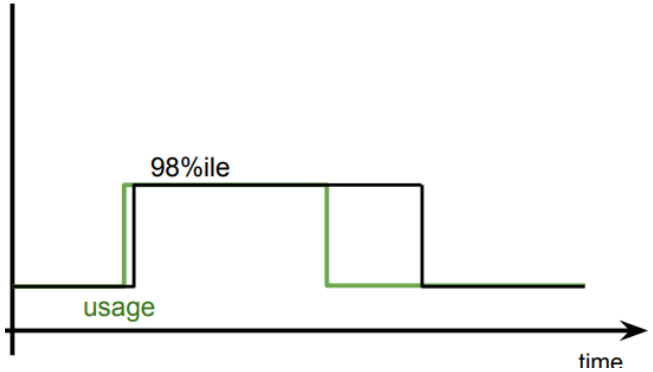
- 滑动窗口预测
  - 滑动窗口算法的作用就是根据一个窗口期内的历史数据，加权预测资源的合理limits
- 增强学习预测（辅助，未实现）
  - 提高预测的准确率
  - 引入先验

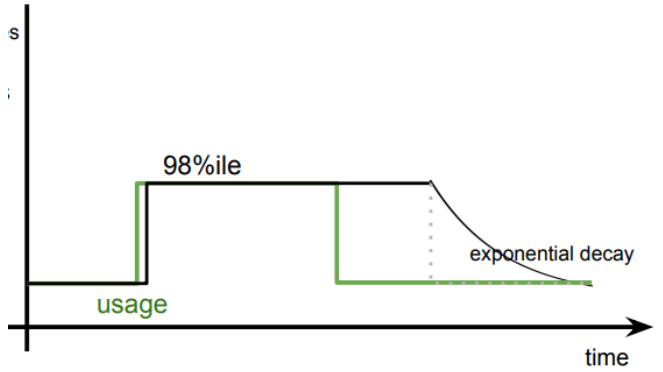
滑动窗口

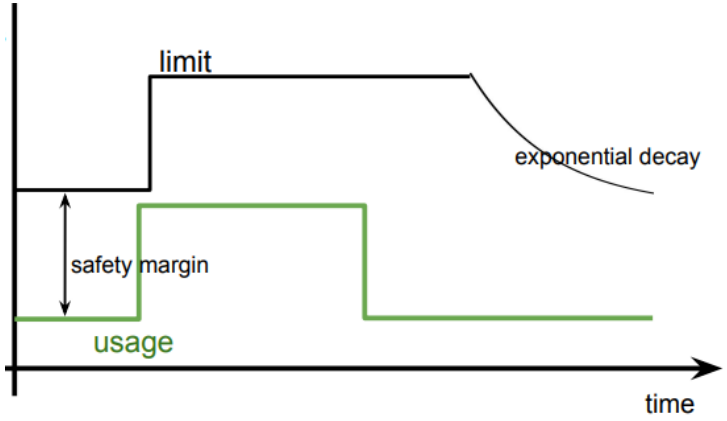
- 指数衰减
  - 历史数据的权重， 越久的数据权重越低
  - 半衰期：CPU=?h, MEM=?h
- 峰值预测=>Upper Bound
  - 取最近N个样本的最大值
- 均值预测=>Trend
  - 历史数据的加权平均
- J-%ile 分位值预测 => Lower Bound
  - 所有历史数据的j-%ile分位加权平均

概念实现

- 指数衰减采样（半衰期）
- 样本统计计算（95%ile）
- 安全边界





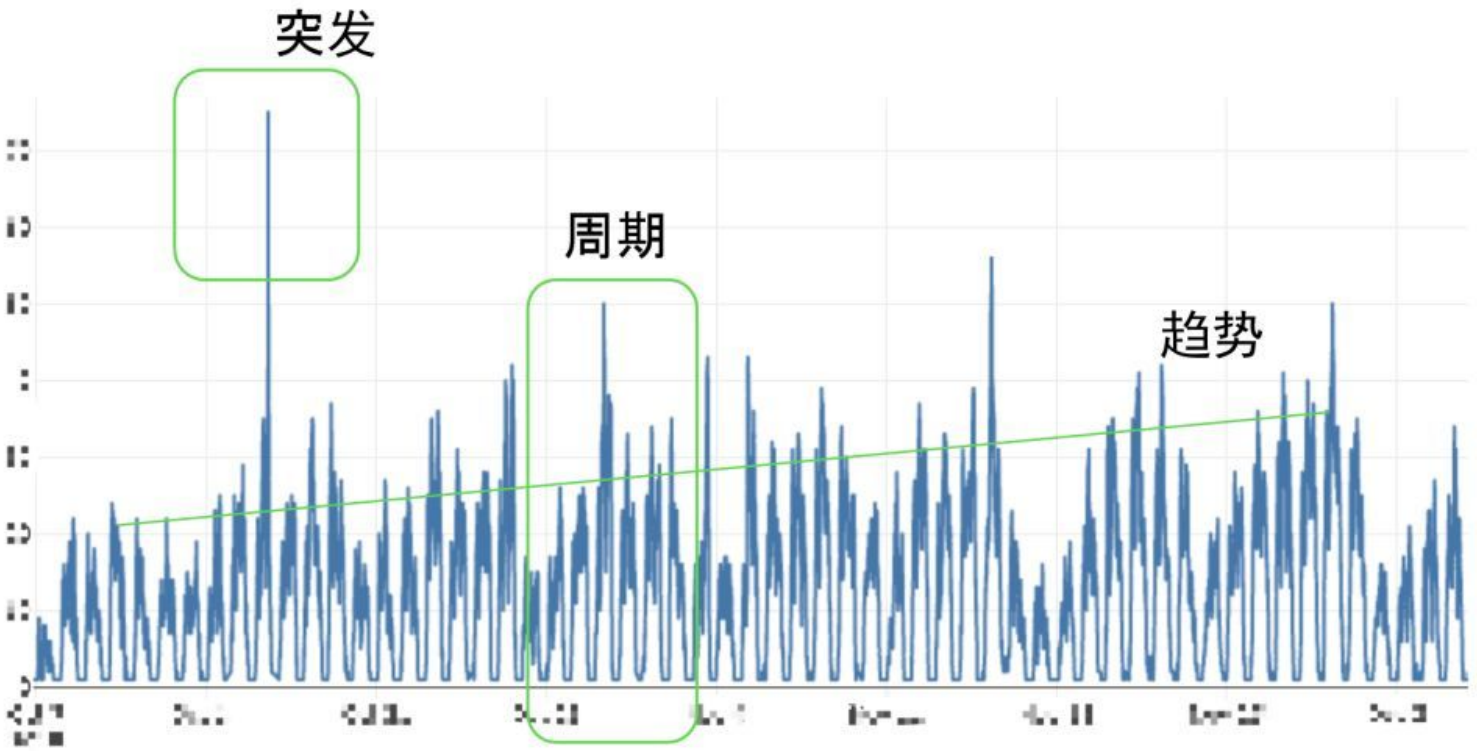


实现

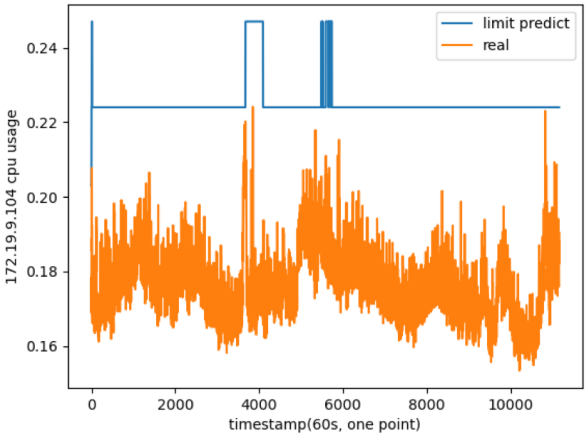
- 基于cli的模式
- 离线处理数据

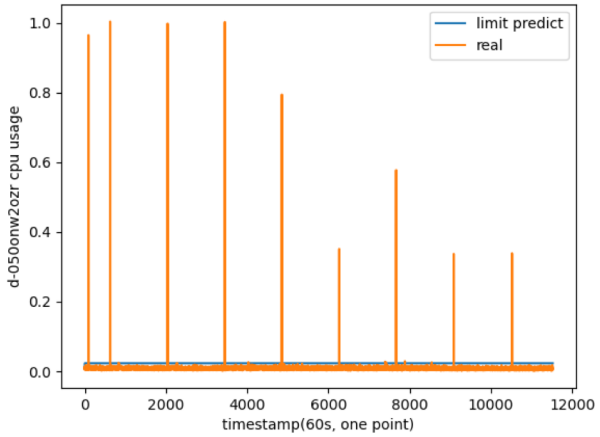
```
[root@A04-R08-I137-I205-9113C72 cloud-prophet]# ./autopilot --help
Usage of ./autopilot:
  -cpu-histogram-decay-half-life duration
        CPU利用率权重减半的周期，半衰期. (default 24h0m0s)
  -csv-file string
        指标数据来自监控系统，csv文件名称 (default "unkowfile")
  -element-id string
        资源利用率预测的实例ID (default "unknow")
  -recommendation-margin-fraction float
        预测的安全边缘余量，百分比 (default 0.15)
  -runonce-timeout duration
        一次运行的最大超时时间 (default 1h0m0s)
  -sample-second-interval int
        样本的采样间隔，单位(秒)，整型 (default 60)
  -target-cpu-percentile float
        cpu预测采用的分位值，百分比 (default 0.9)
```

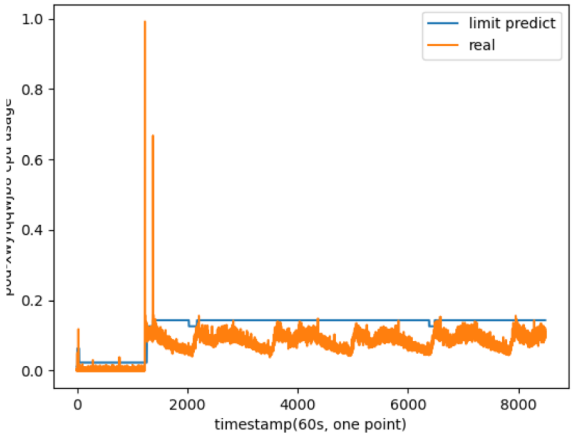
数据特征

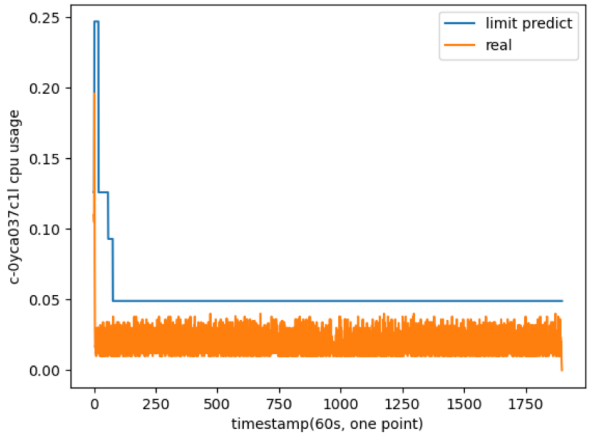


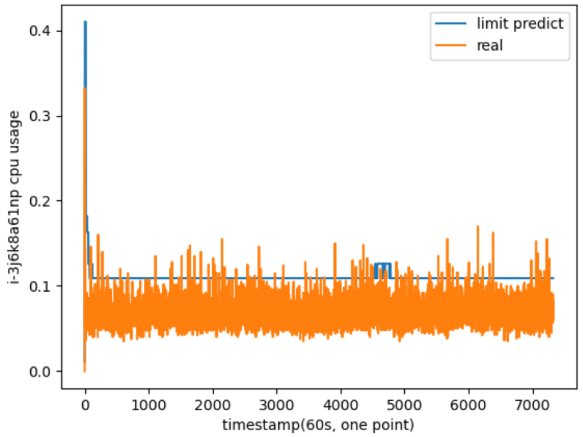
实验结果（1）





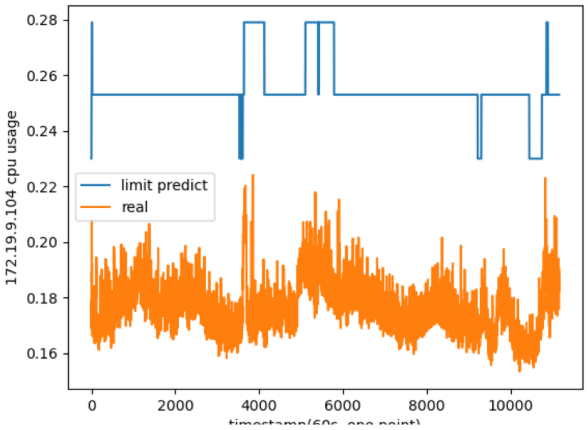


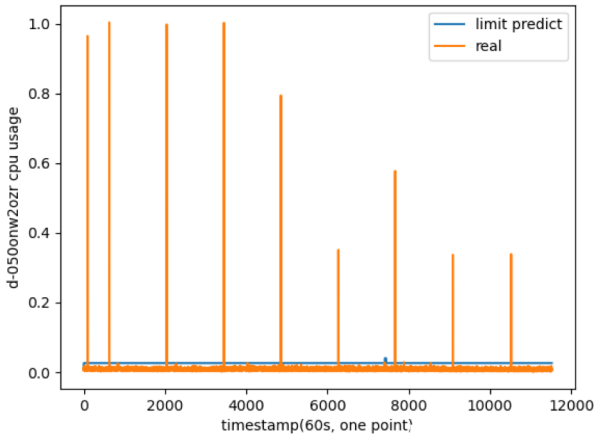


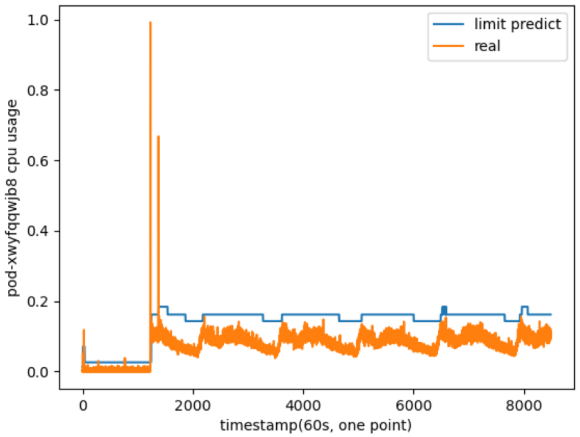


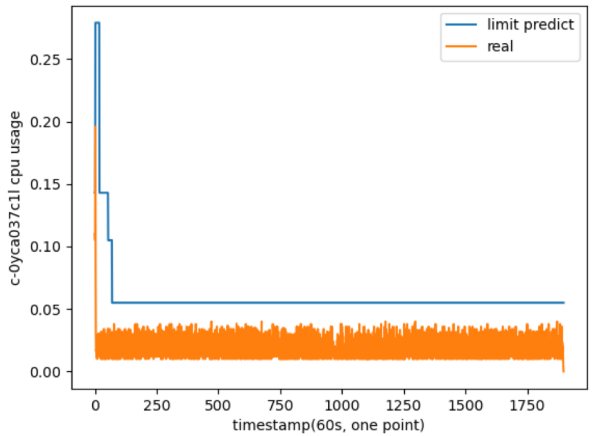
--sample-second-interval 60 --target-cpu-percentile 0.95 --cpu-histogram-decay-half-life 12h0m0s --recommendation-margin-fraction 0.15

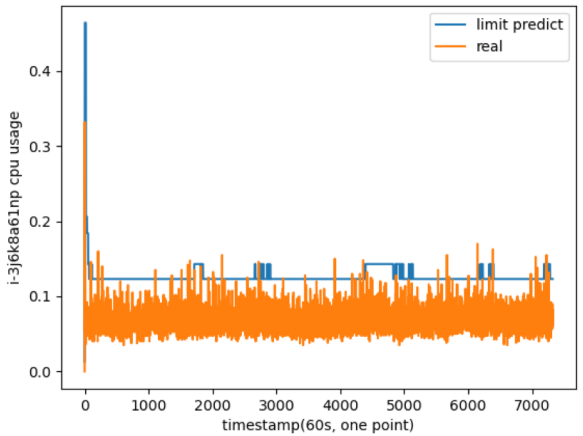
实验结果（2）





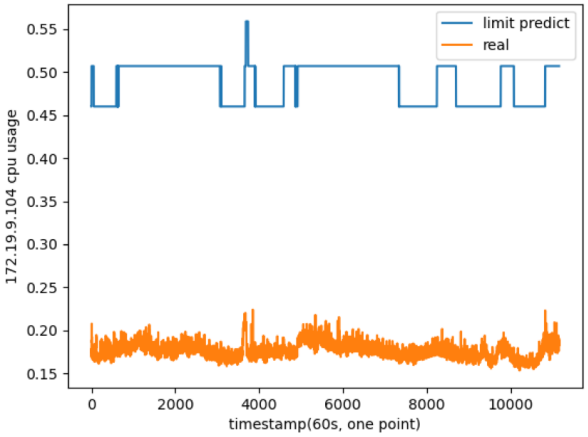


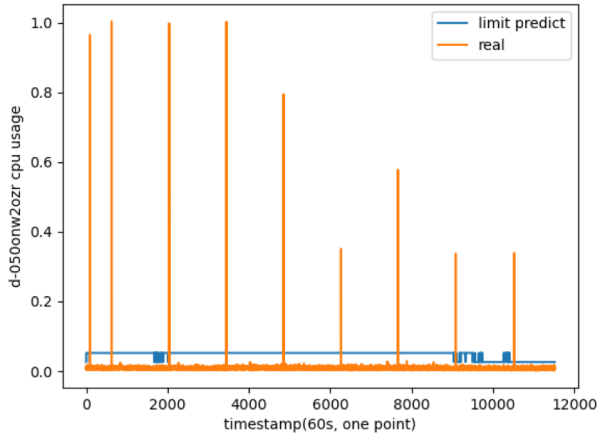


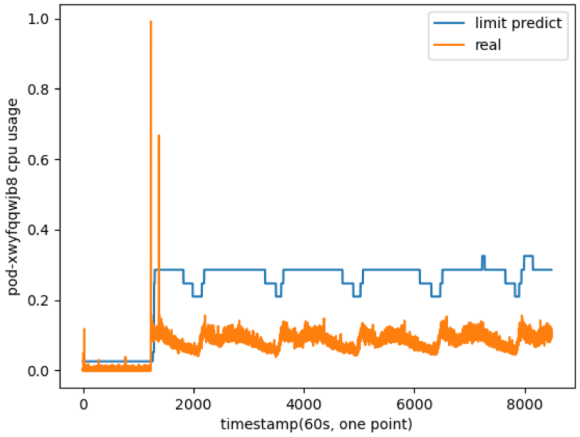


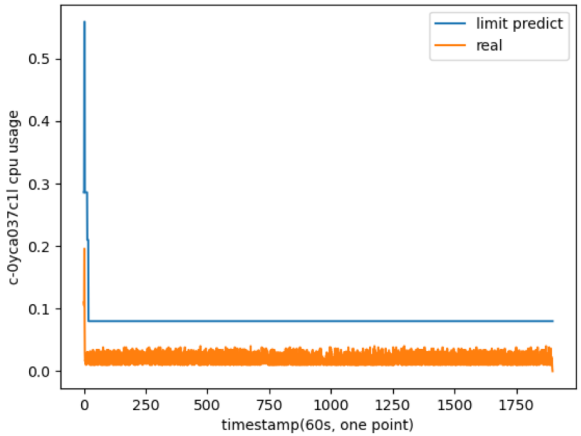
--sample-second-interval 60 --target-cpu-percentile 0.95 --cpu-histogram-decay-half-life 3h0m0s --recommendation-margin-fraction 0.30

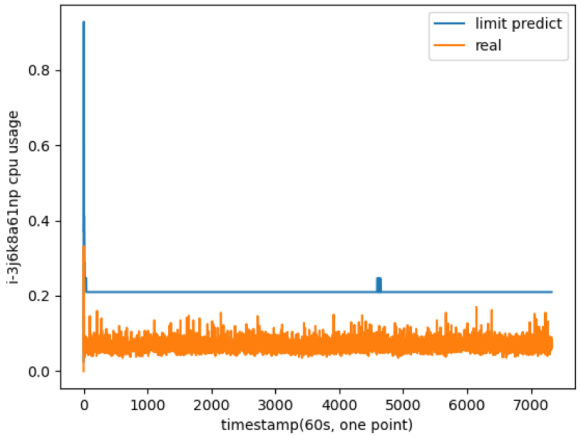
实验结果（3）











--sample-second-interval 60 --target-cpu-percentile 0.8 --cpu-histogram-decay-half-life 3h0m0s --recommendation-margin-fraction 1.6

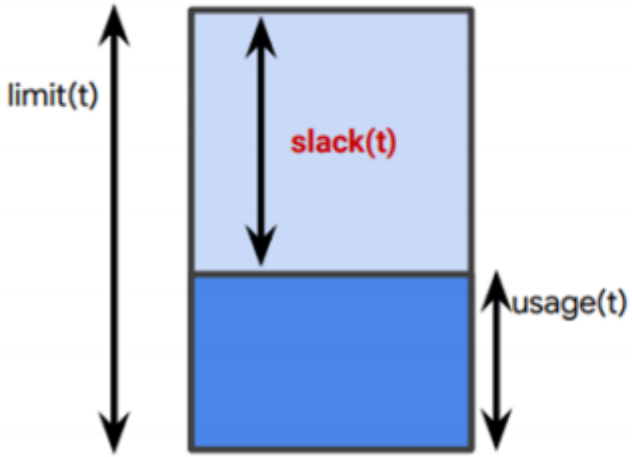
结果分析

- 选取的样本具有一定的代表性
  - 涵盖了所有jdcloud资源实例（host,vm,nc,pod,docker）
  - 突增的资源消耗，无明显规律的资源消耗，见（1,2）
  - 比较固定的时间变化模式，见（3）
  - 比较稳定的资源利用，见（4, 5）
- 方法具有
  - 处理流式时序数据的能力
  - 接近实时
  - 准确性（可用）
  - 参数化，可调配

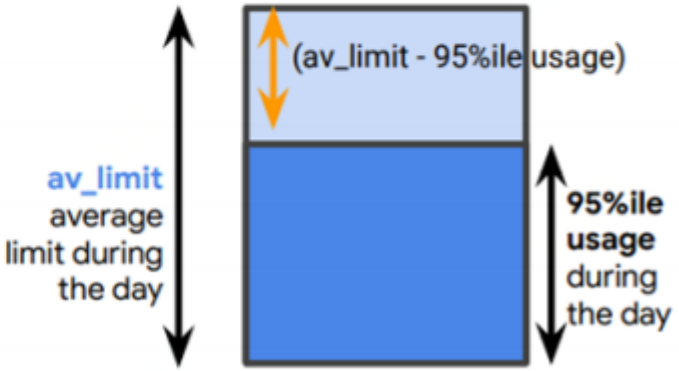
参考测评（google）

absolute slack:
∫ slack(t) dt = ∫ limit(t) dt - ∫ usage(t) dt

unit: capacity of a single (largish) machine



relative slack:
(av_limit - 95%ile usage) / (av_limit)



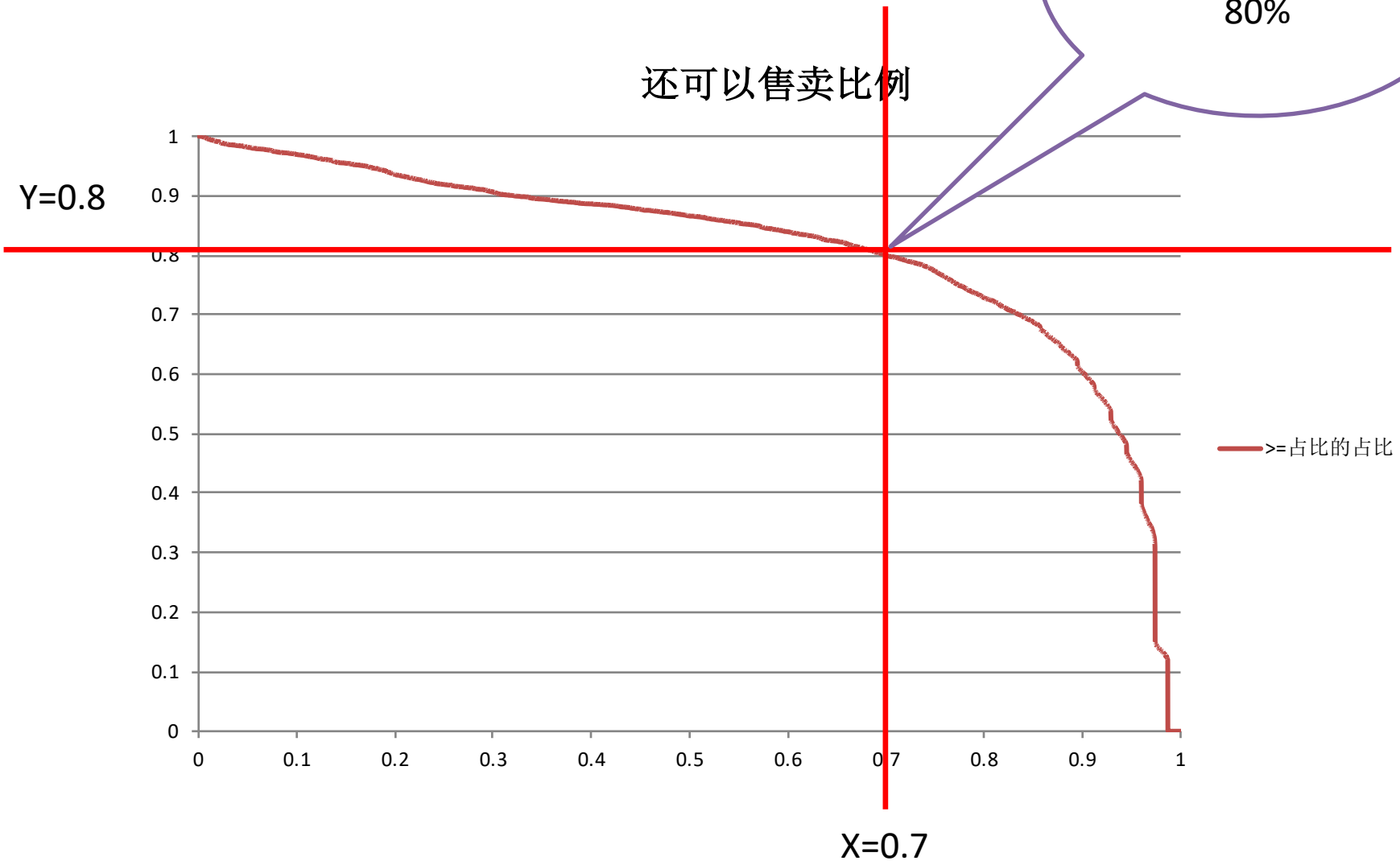
思想可以借鉴，实施不太匹配
我们的指标数据已经百分比规格化，而不是按时间片模式

综合测评

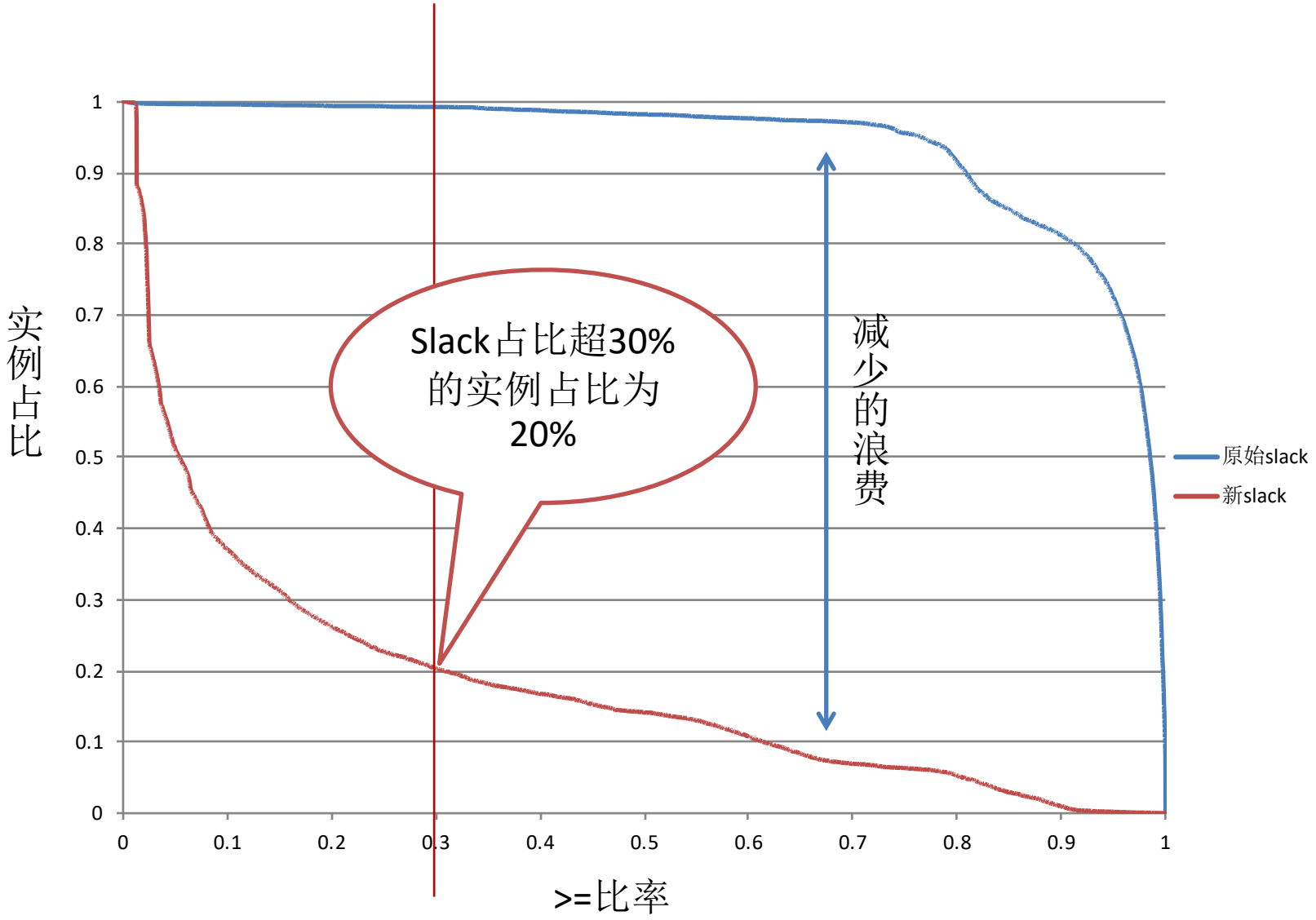
- 给定一个线上实例
  - 对应唯一规格
  - limit为规格的定义，静态的
- 引入predicted limit
  - 没使用的资源slack
    - (predicted) limit – used
    - 越小越好：表征已分配但没使用到的资源
  - 可以卖的
    - 1-(predicted) limit
    - 越大越好：超出原本的售卖能力(超卖)
- 样本源
  - 部分华北general_2实例（数目~7000）
  - 最大采集近一周的数据
  - cpu利用率原始值，10s间隔

超卖能力

可超卖>70%的
已售出规格占
80%



资源浪费（slack）


实例占比
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
>=比率
Slack占比超30%
的实例占比为
20%
减少的浪费
原始slack
新slack

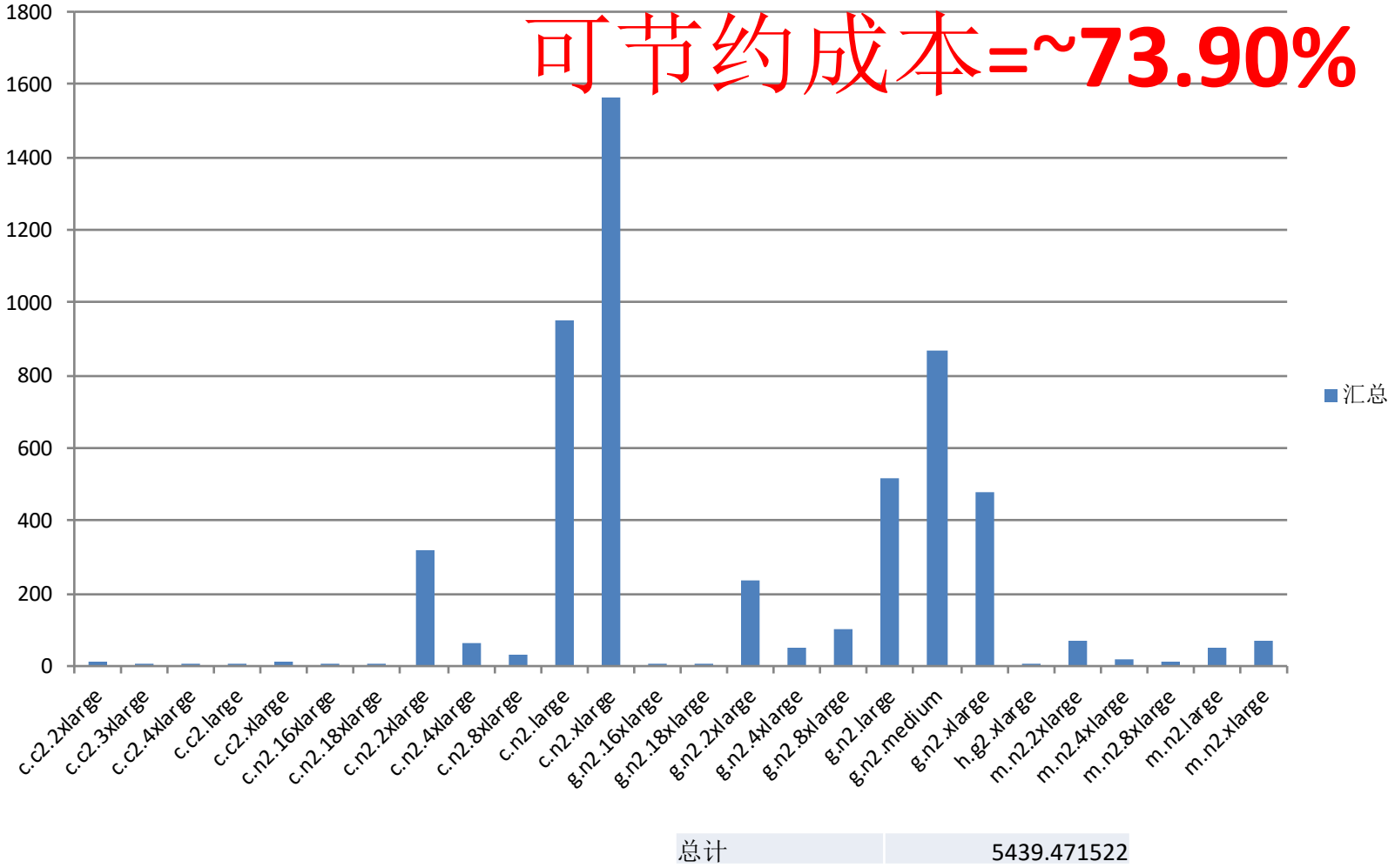
原地增收（理想情况）

- 不增加新机器的原地增收
- 按照上面~7000实例的规格
- 按照predicted limit划分
- 新增可售卖的按规格加权CPU资源
  - 新增可售卖

$$\sum_{n=1}^{\sim 7000} \left(instance_{n_{flavor}} \right) * (1 - PredictLimit(n))$$

原地增收

新增可售卖规格数目/~7000实例



总计	5439.471522

其他

- 思路：从卖规格到卖QoS
- 超卖：挖掘超出原有售卖能力的售卖
- 参考模型：
  - 从同一物理机上的allocations到allocation-set
  - 资源 recommender + updater
- 前提条件：
  - 数据规模
  - 预测
    - 实时性
    - 准确性

参考

- “Autopilot: workload autoscaling at Google”
  - https://dl.acm.org/doi/pdf/10.1145/3342195.3387524
- “预测实验对比”
  - https://git.jd.com/iaas-sdn/jvirt-doc/blob/master/cluster/2020V8/资源利用率提升/数据驱动/预测实验.md

反碎片化实践方案

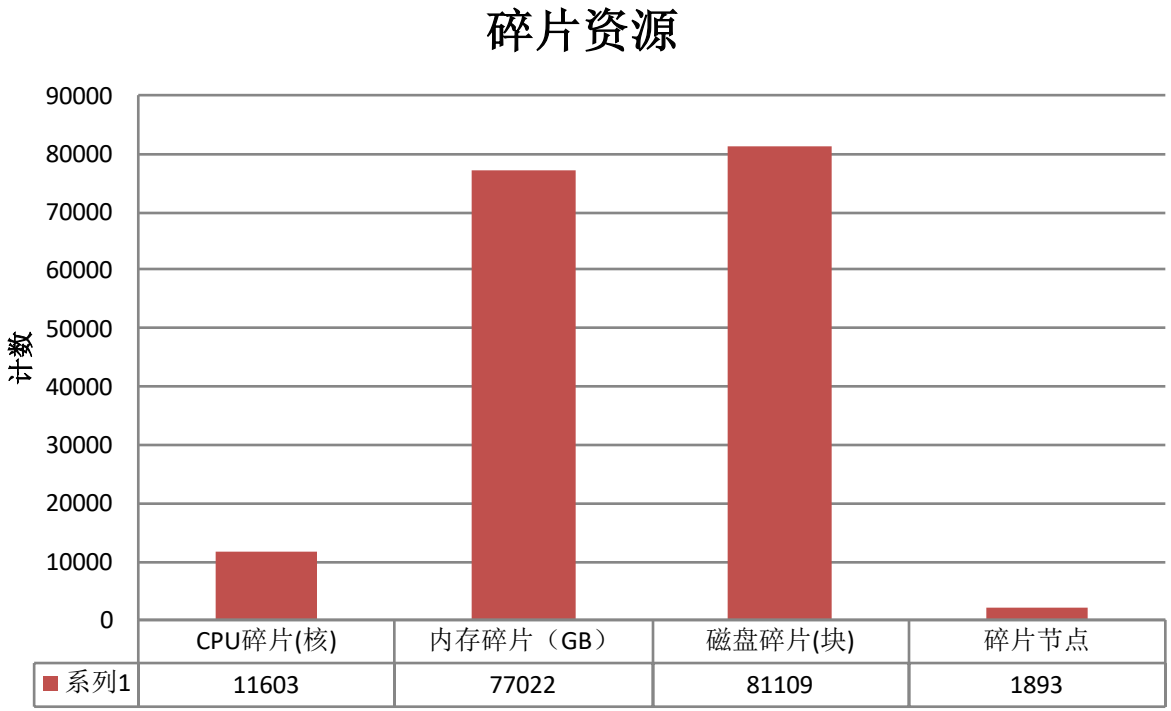
- 问题&背景
- 模型&算法
- 评估

问题描述

- 有资源无法调度
  - 节点无MEM，其他维度资源都是碎片
  - 节点无CPU，其他维度资源都是碎片
  - 暂不考虑使能的影响
    - ~~节点/服务disable~~
    - ~~节点故障~~

现状

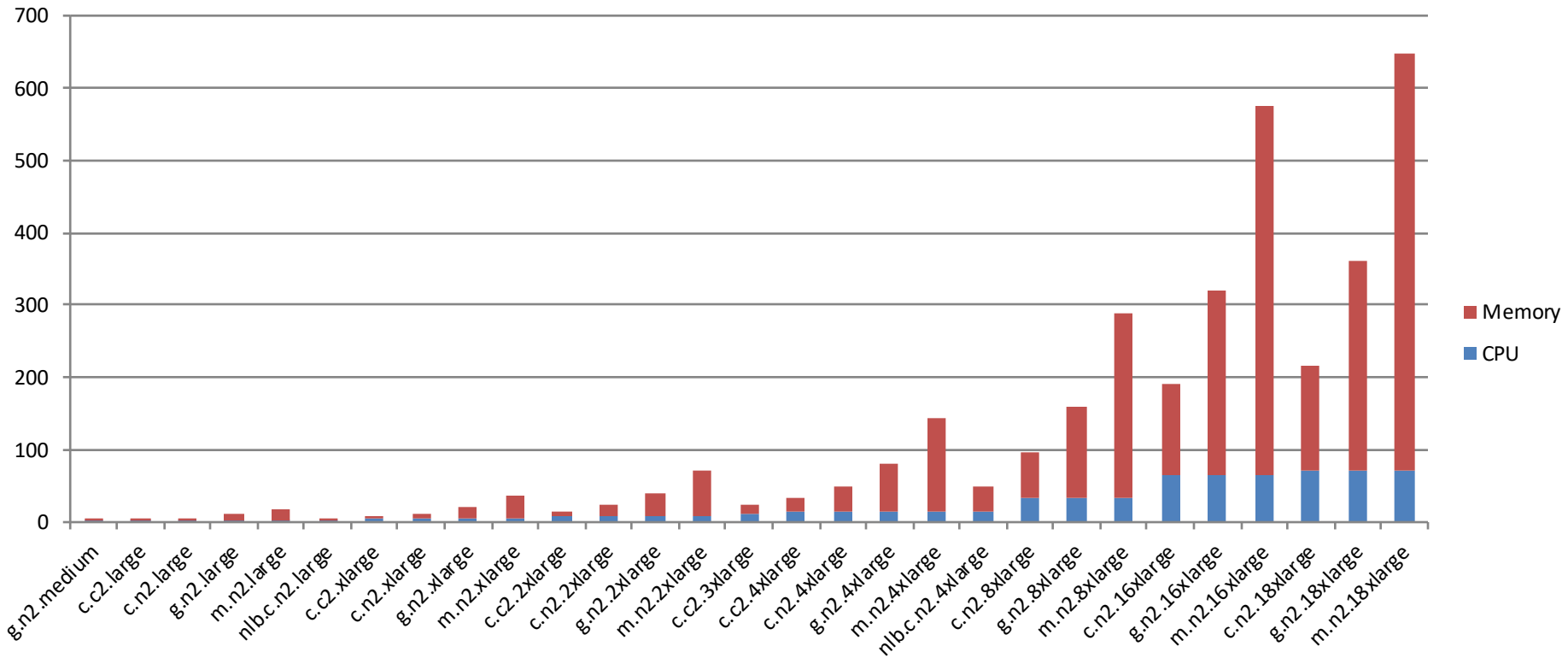
- 样本
  - 华北通用二代机器~6000+台线上计算节点
  - 其中384台节点关闭调度



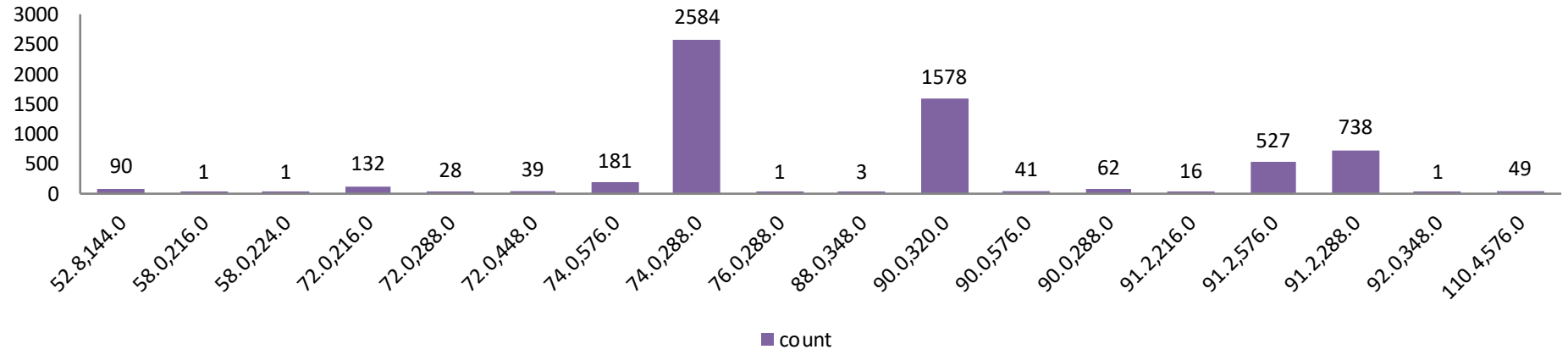
数据集

- 规格分布：~20+ flavors
- 节点分布：~6000+ hosts
- 实例分布：~50000+ instances (vm, nc, pod)

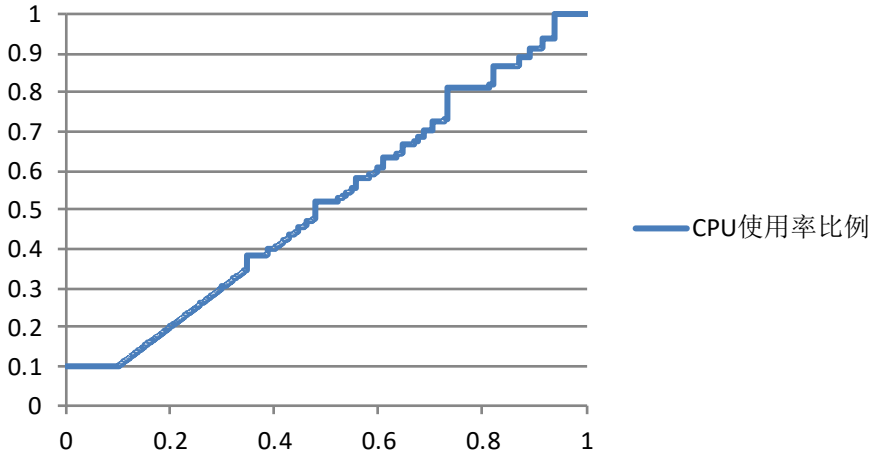
規格


700
600
500
400
300
200
100
0
Memory
CPU
g.n2.medium
c.c2.large
c.n2.large
g.n2.large
m.n2.large
nlb.c.n2.large
c.c2.xlarge
c.n2.xlarge
g.n2.xlarge
m.n2.xlarge
c.c2.2xlarge
c.n2.2xlarge
g.n2.2xlarge
m.n2.2xlarge
c.c2.3xlarge
c.c2.4xlarge
c.n2.4xlarge
g.n2.4xlarge
m.n2.4xlarge
nlb.c.n2.4xlarge
c.n2.8xlarge
g.n2.8xlarge
m.n2.8xlarge
c.n2.16xlarge
g.n2.16xlarge
m.n2.16xlarge
c.n2.18xlarge
g.n2.18xlarge
m.n2.18xlarge

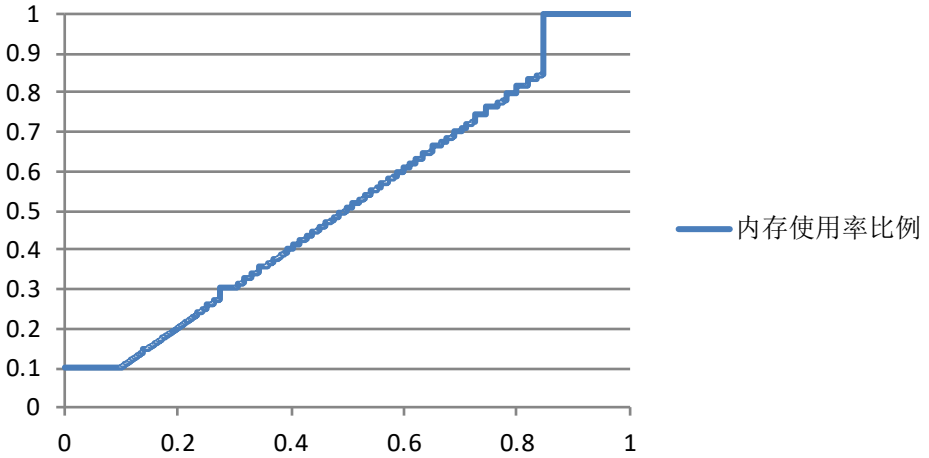
节点



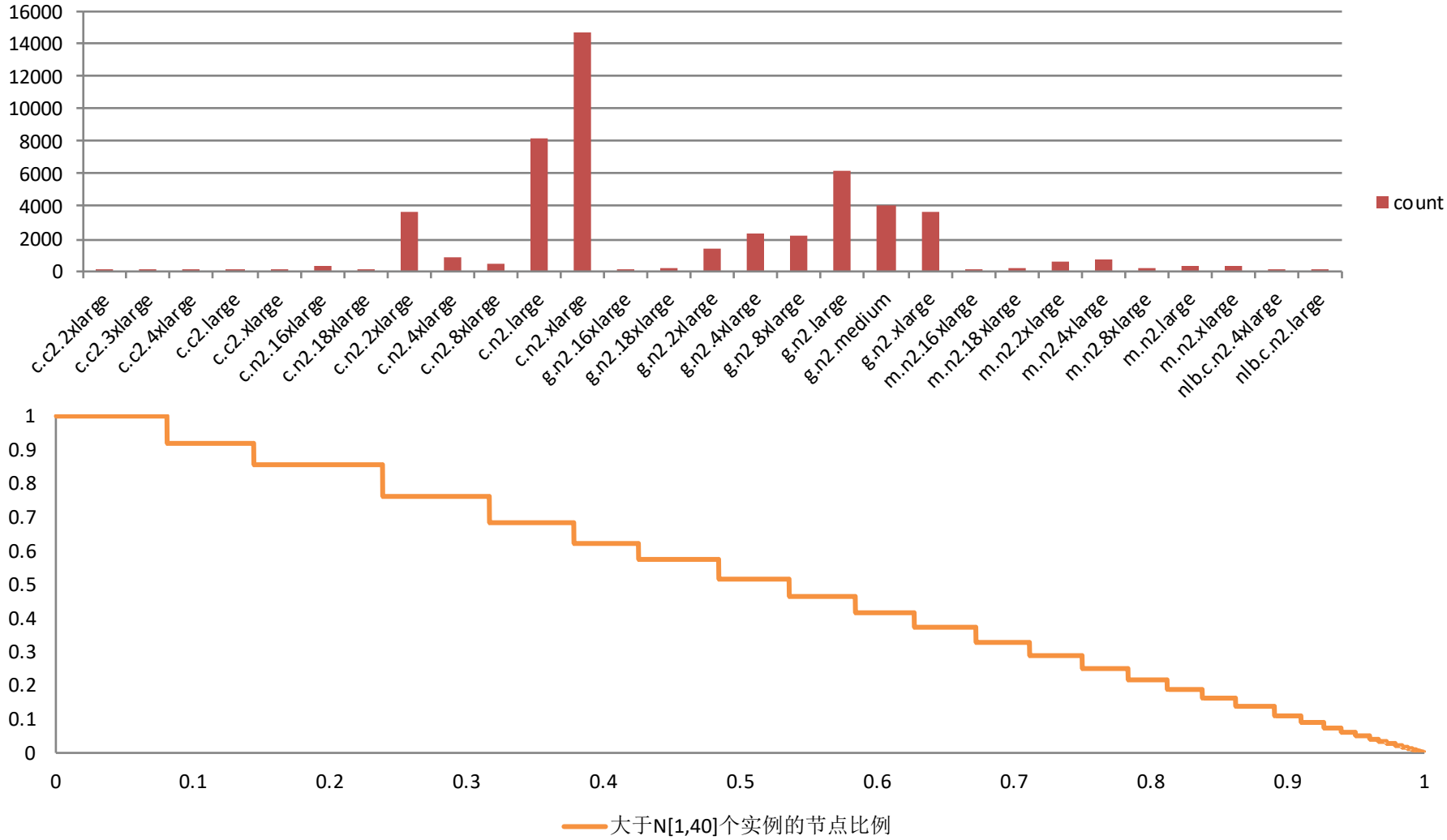
CPU使用率比例



内存使用率比例



实例


16000
14000
12000
10000
8000
6000
4000
2000
0
count
c.c2.2xlarge
c.c2.3xlarge
c.c2.4xlarge
c.c2.large
c.c2.xlarge
c.n2.16xlarge
c.n2.18xlarge
c.n2.2xlarge
c.n2.4xlarge
c.n2.8xlarge
c.n2.large
c.n2.xlarge
g.n2.16xlarge
g.n2.18xlarge
g.n2.2xlarge
g.n2.4xlarge
g.n2.8xlarge
g.n2.large
g.n2.medium
g.n2.xlarge
m.n2.16xlarge
m.n2.18xlarge
m.n2.2xlarge
m.n2.4xlarge
m.n2.8xlarge
m.n2.large
m.n2.xlarge
nlb.c.n2.4xlarge
nlb.c.n2.large
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
大于N[1,40]个实例的节点比例

Bin Packing

主流算法	描述
Next fit	满足则装当前打开的1个bin, 不满足开1个新bin放入
Next k fit	打开上次k个bin，找一个满足，不满足开1个新bin放入
First fit	从已有bin找第一个满足的bin，不满足开1个新bin放入
Best fit	从已有bin找最合适的bin，不满足开1个新bin放入
OPT fit	看到所有bin，所有package, 用最少的bin装填，必然碎片最少

装箱参数	资源分派、放置	备注
#bins, capacity	#hosts	bin一般都一样，节点有差异
#packages, size	#flavors	package有长宽高，并且可旋转

综合评估

- 调度评估
  - 最优化调度
- 综合测评指标
  - 碎片程度，越少越好（计划任务所需机器少）
  - 节点状态，越均衡利用越好
  - 备机需求，越少越好
  - 迁移代价，[迁移次数，涉及节点，候选实例]

调度评估

- Classic Definition
  - A packing heuristic
    - Resource allocation demand vector “<” machine resource vector
    - “<”: alignment score (AS)
  - “AS” works because:
    - No over-allocation
    - Bigger balls get bigger scores
    - Abundant resources used first
- NP hard
  - 最优化方法、整数规划、约束规划、元启发式方法
  - 简化为动态规划

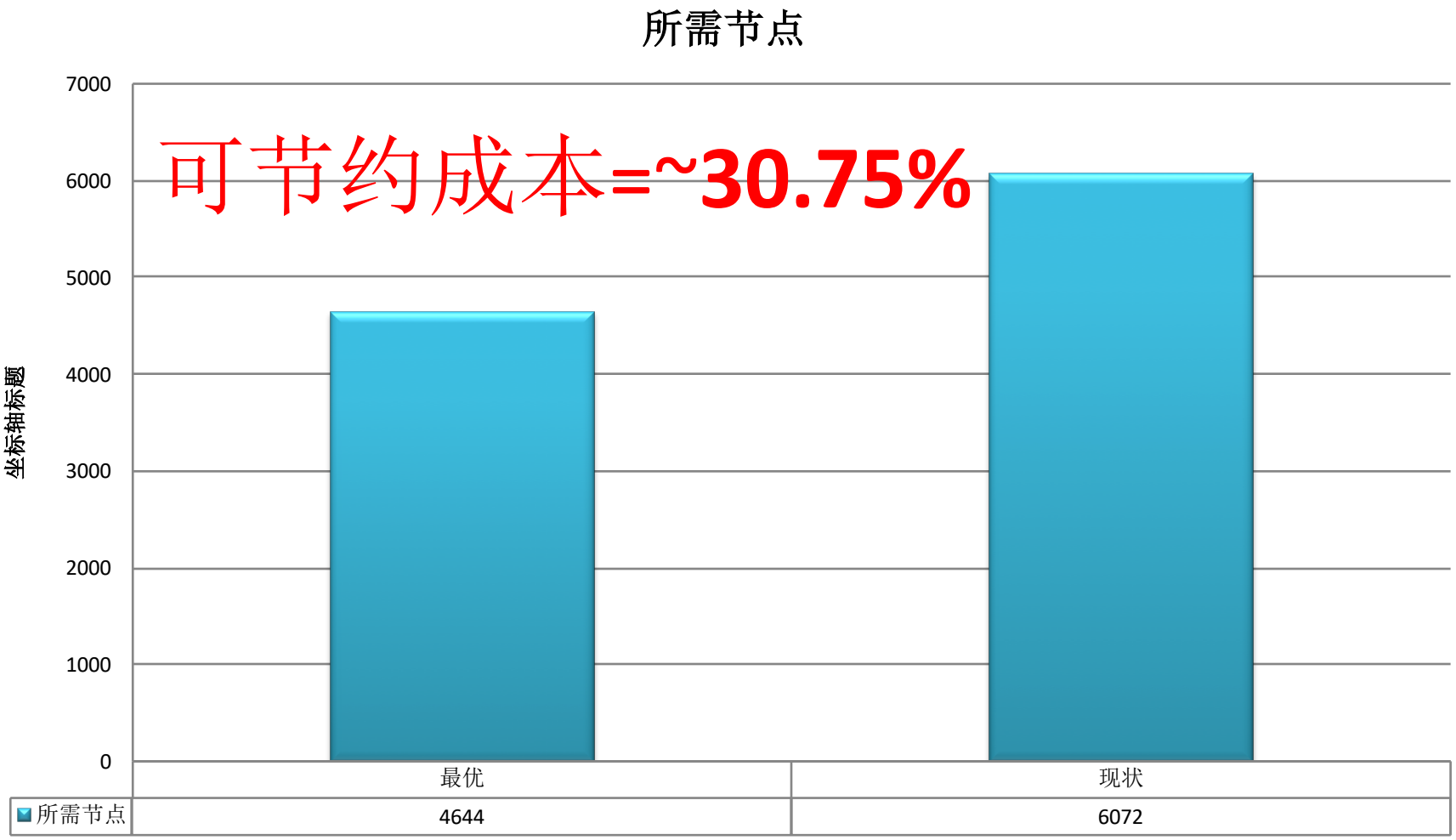
全局装箱任务

bins			packages			
capacity	count		规格	CPU	Memory	count
52.8,144.0	90		g.n2.medium	1	4	4108
58.0,216.0	1		c.c2.large	2	2	65
58.0,224.0	1		c.n2.large	2	4	8136
72.0,216.0	132		g.n2.large	2	8	6175
72.0,288.0	28		m.n2.large	2	16	288
72.0,448.0	39		nlb.c.n2.large	2	4	10
74.0,576.0	181		c.c2.xlarge	4	4	77
74.0,288.0	2584		c.n2.xlarge	4	8	14741
76.0,288.0	1		g.n2.xlarge	4	16	3626
88.0,348.0	3		m.n2.xlarge	4	32	297
90.0,320.0	1578		c.c2.2xlarge	8	8	97
90.0,576.0	41		c.n2.2xlarge	8	16	3602
90.0,288.0	62		g.n2.2xlarge	8	32	1418
91.2,216.0	16		m.n2.2xlarge	8	64	646
91.2,576.0	527		c.c2.3xlarge	12	12	2
91.2,288.0	738		c.c2.4xlarge	16	16	93
92.0,348.0	1		c.n2.4xlarge	16	32	835
110.4,576.0	49		g.n2.4xlarge	16	64	2339
总节点数	6072		m.n2.4xlarge	16	128	756
			nlb.c.n2.4xlarge	16	32	42
			c.n2.8xlarge	32	64	427
			g.n2.8xlarge	32	128	2175
			m.n2.8xlarge	32	256	170
			c.n2.16xlarge	64	128	353
			g.n2.16xlarge	64	256	46
			m.n2.16xlarge	64	512	10
			c.n2.18xlarge	72	144	15
			g.n2.18xlarge	72	288	178
			m.n2.18xlarge	72	576	207
					总实例数	50934

最优化方案（理想情况）

```
============================ Problem Statistics ============================
         | PMs          | Jobs         | VMs          | Mappings     | Used PMs
 Number  |         6072 |        50934 |        50934 |            0 |            0
 ----------------------------------------------------------------------------
 ----------------------------------------------------------------------------
                            | CPU                  | RAM
 ---------------------------------------------------------------------------
 Total Capacity             |               497417 |              2011784
 Minimum Capacity           |                   52 |                   144
 Maximum Capacity           |                  110 |                   576
 Average Capacity           |                81.92 |                331.32
 ---------------------------------------------------------------------------
 Total Requirements         |               362848 |              1294062
 Minimum Requirements       |                    1 |                     2
 Maximum Requirements       |                   72 |                   576
 Average Requirements       |                 7.12 |                 25.41
 Usage Percentile           | 0.7294644132000000 | 0.6432410239000000
============================================================================
Solution using 4644 PMs found
Elapsed time: 1036.597843154 seconds
SUCCESS
```

降本


所需节点
可节约成本=~30.75%
坐标轴标题
7000
6000
5000
4000
3000
2000
1000
0
最优
现状
所需节点
4644
6072

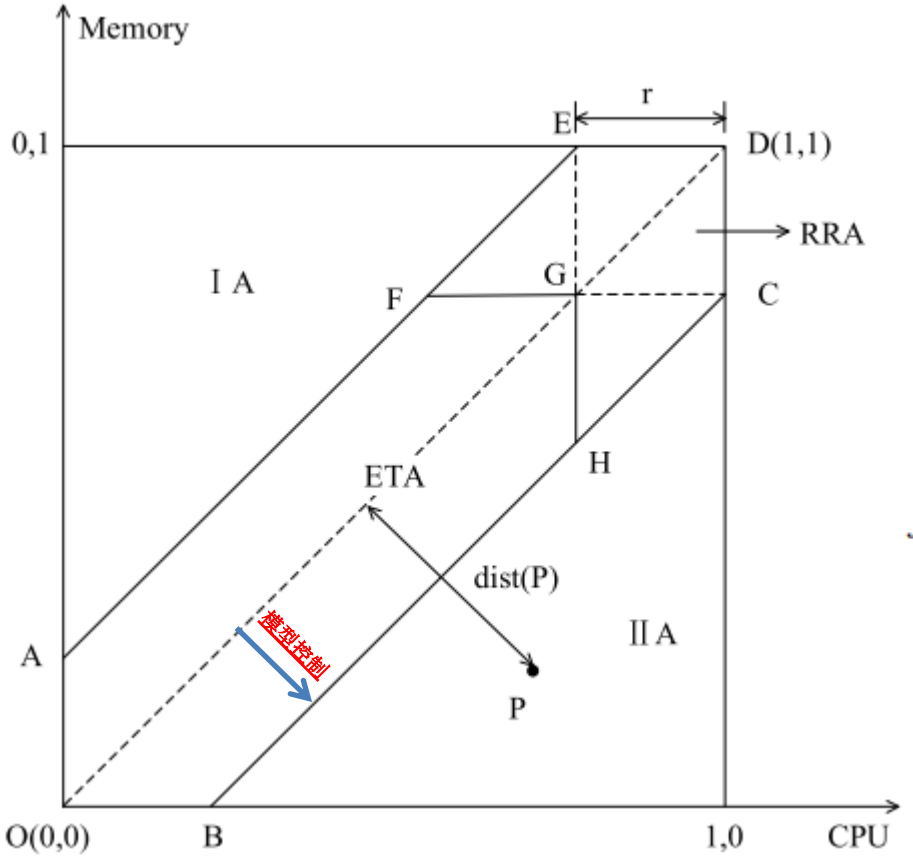
根因分析

- 规格不按比例齐整
- 节点上线有时间线
- 其他调度参数
  - 指定节点
  - Tag
  - 亲和性, HG
- 时序请求无法最优
  - 请求有先有后
  - 删除+创建

解决方法

- 节点分类
  - 空闲机器
  - 满载机器
  - 偏科机器
- 迁移计划➔再平衡，利用率优先
  - 调度
  - 效率
  - 目标

节点如何，为什么分类

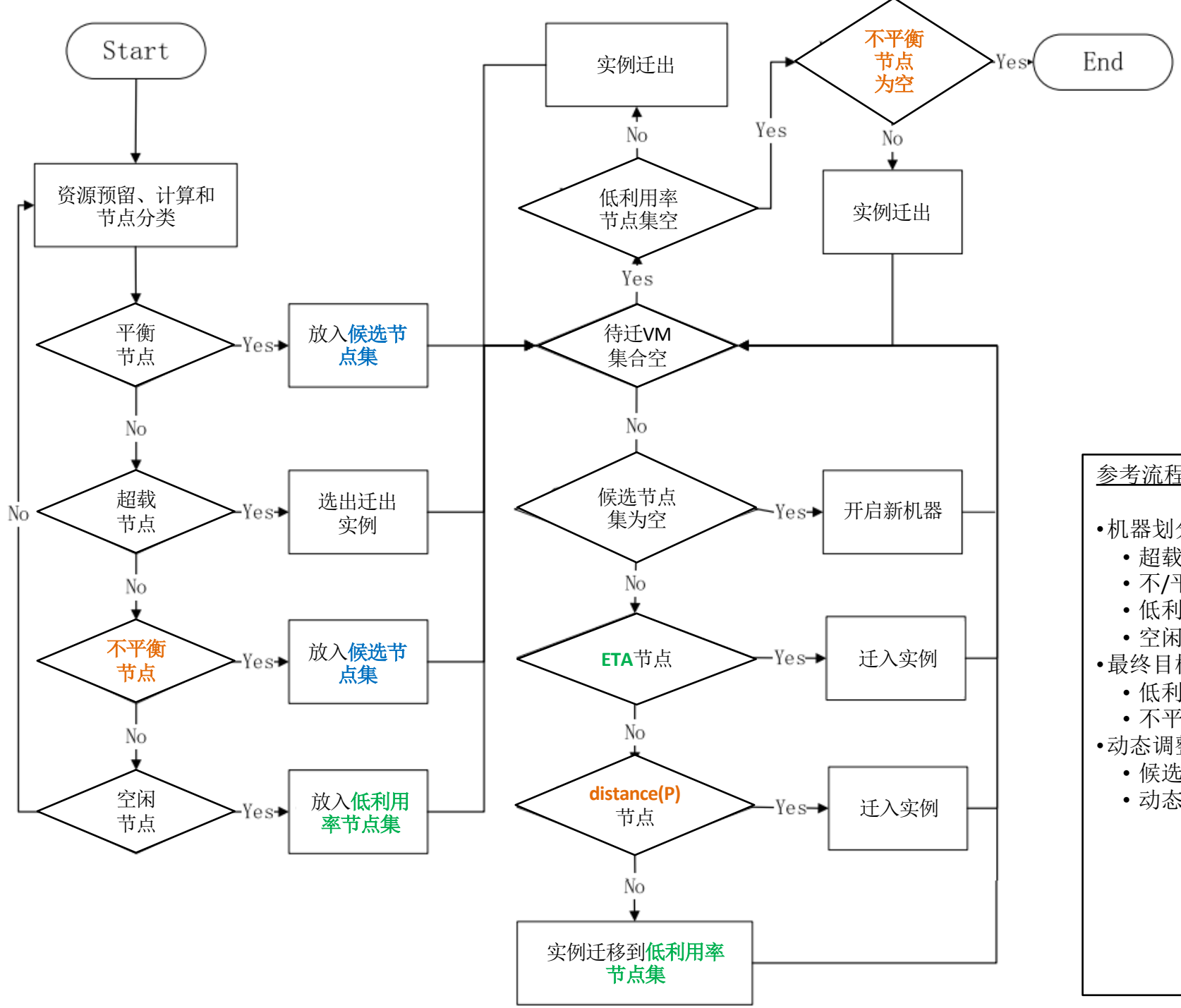


$$f(x,y)=\begin{cases} \underline{\text{（饱和，过载）}} \\ RRA, & (1-x)\le r \& (1-y)\le r \\ \underline{\text{（平衡）}} \\ ETA, & dist(p)\le \dfrac{\sqrt{2}}{2}r \\ & \&(1-x)>r\&(1-y)>r \\ \underline{\text{（偏科）}} \\ IA\ or\ IIA, & otherwise \end{cases}$$

- 基于资源利用分布对节点建模
  - 通过资源分布划分节点类型
  - 根据节点类型，判定节点迁移状态、迁移调度

物理意义

- RRA: Resource Reserved Area, CDEFGH
  - 应对负载突增
  - 保证SLA
- ETA: Equilibrium Tolerance Area, AOBHGF
  - 合理利用率
- Area I & Area II: IA, IIA
  - 不平衡区域
  - Dimension Rate



参考流程：

- 机器划分
  - 超载
  - 不/平衡(ETA)
  - 低利用率
  - 空闲
- 最终目标状态
  - 低利用率节点为空
  - 不平衡节点为空
- 动态调整
  - 候选机器集合
  - 动态实例集合

应用场景

- 动态备机遴选
  - 去掉常规备机方案
    - 664 + 21 + 58 + 44
    - 近1万/天
  - 增加售卖
- 预选目标机
  - 支持运维迁移计划
- 增加可售卖
  - 去碎片化
- 预测性维护
  - 兼顾资源利用

深入方向

- 考虑
  - 能耗（Energy consumption），也是需要钱的
    - 和利用率相关

预测性维护

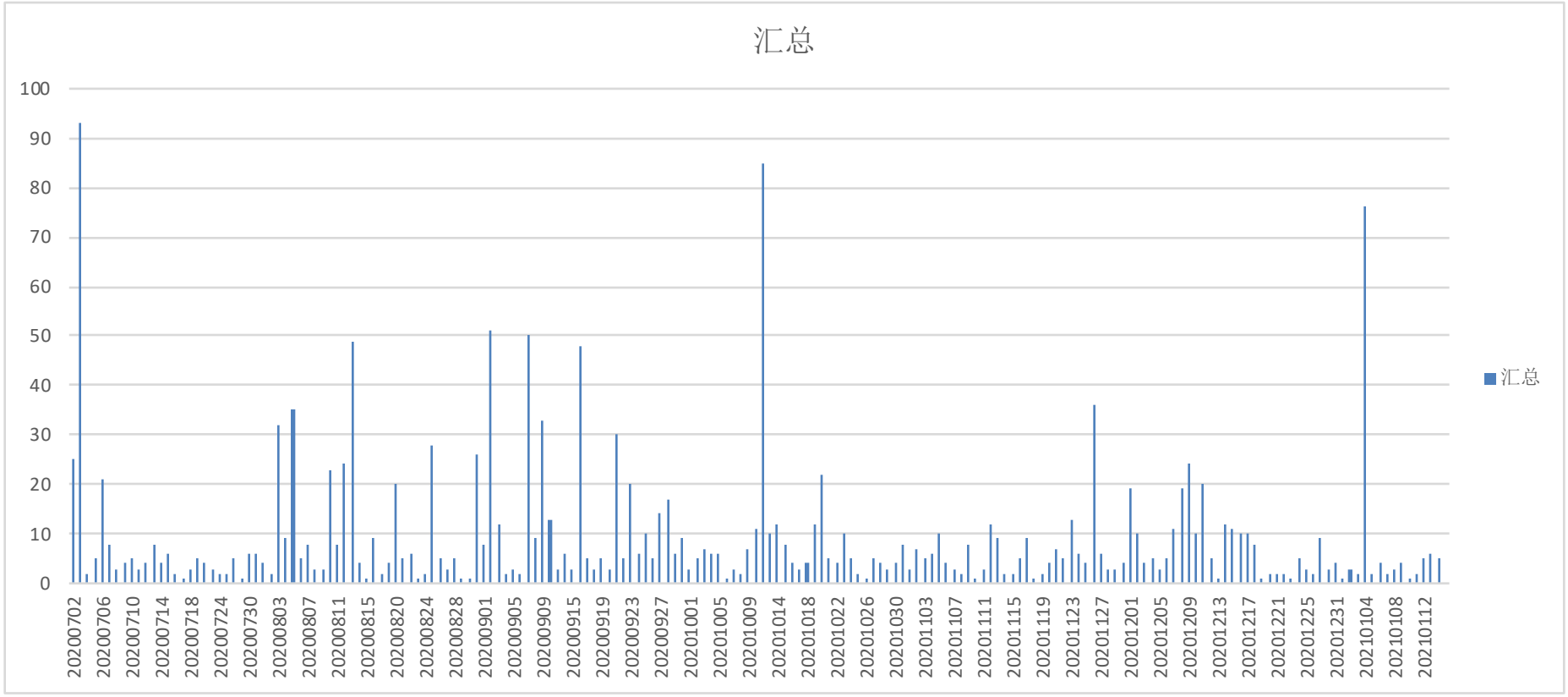
- 事后维护影响SLA
  - 代价大
  - 临时性
  - 缺乏优化目标
- 事前维护
  - 日常Routine
  - 保护SLA

问题

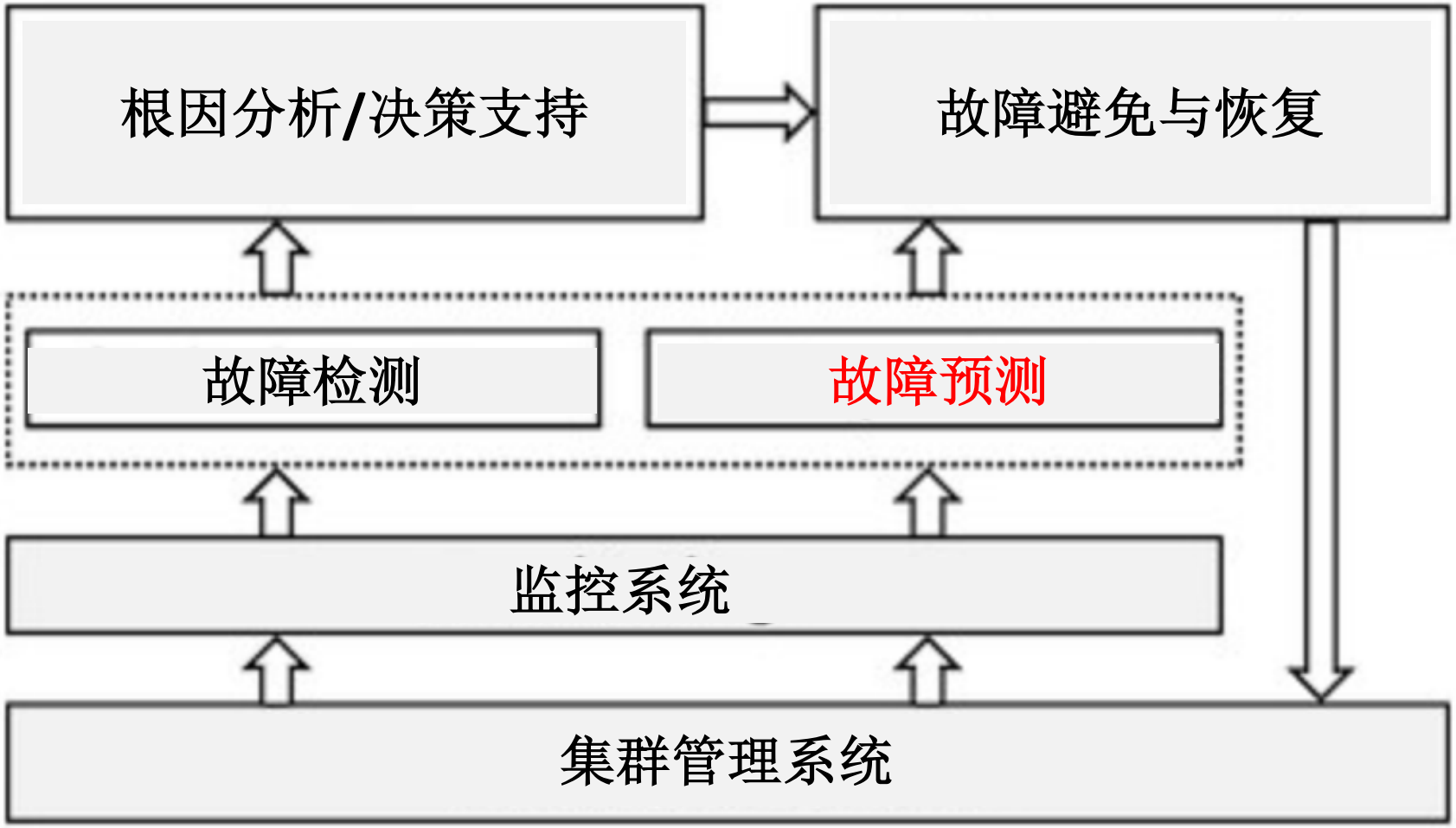
- 连续出现节点故障
- 无法做到故障避免
- 影响业务连续性

现状

- 按时间统计故障机分布
- 按时间统计影响实例分布



框架



数据

- 数据集
  - 正例
    - 收集线上迄今故障节点和故障时间点
    - 以故障点为终止时间往前搜集一周监控指标
      - Cpu，Mem，Network，Disk
      - 历史原因可能采样点比较大
  - 反例
    - 整理中同物理型号和位置节点
    - 近期无故障节点
    - 时间窗口对应
  - 分离出训练集和测试集

方法

- 指标数据滑动窗口
- 节点指标分别训练预测
  - CNN-LSTM
  - 未来的各项指标值
- 基于历史数据训练分类器
  - PCA + SVM
  - 通过指标判定故障还是非故障
- 联合所有预测数据，过分类器
  - 预测故障

提升空间

启发式资源管理

- 现状分析
- 问题

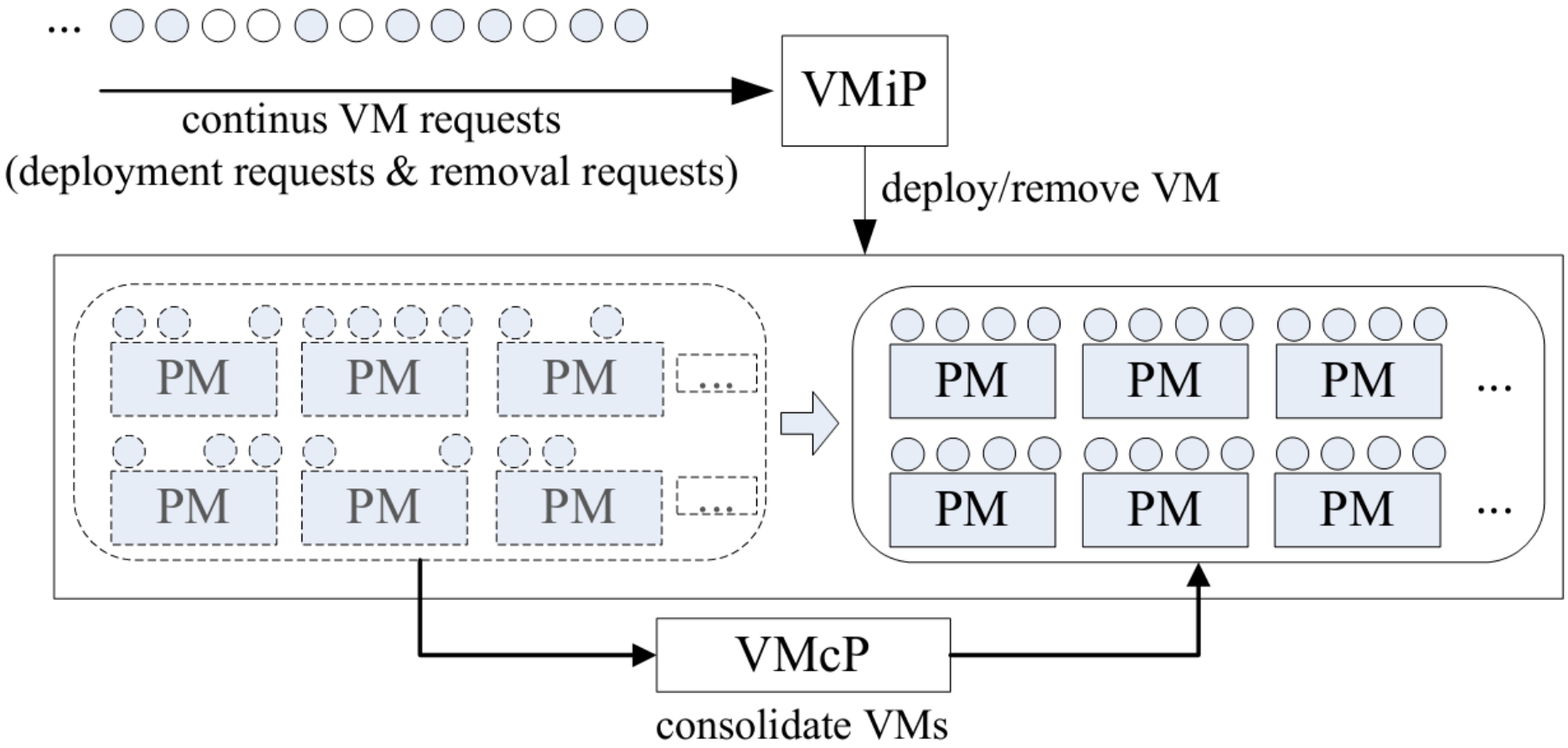
算法

- Vector bin packing
- NP-hard
- Greedy-algorithm
- Worse fit & Best fit
- First fit decreasing (FFD)
  - FFD heuristics
- Best fit decreasing (BFD)
  - BFD heuristics

优化

- Dimension extension
- Power consumption
- Resource wastage
- Server loads
- Inter-VM
- Storage/Network traffic

回顾资源调度



两个场景：连续请求（incremental）, 动态调整(consolidate)

Vector

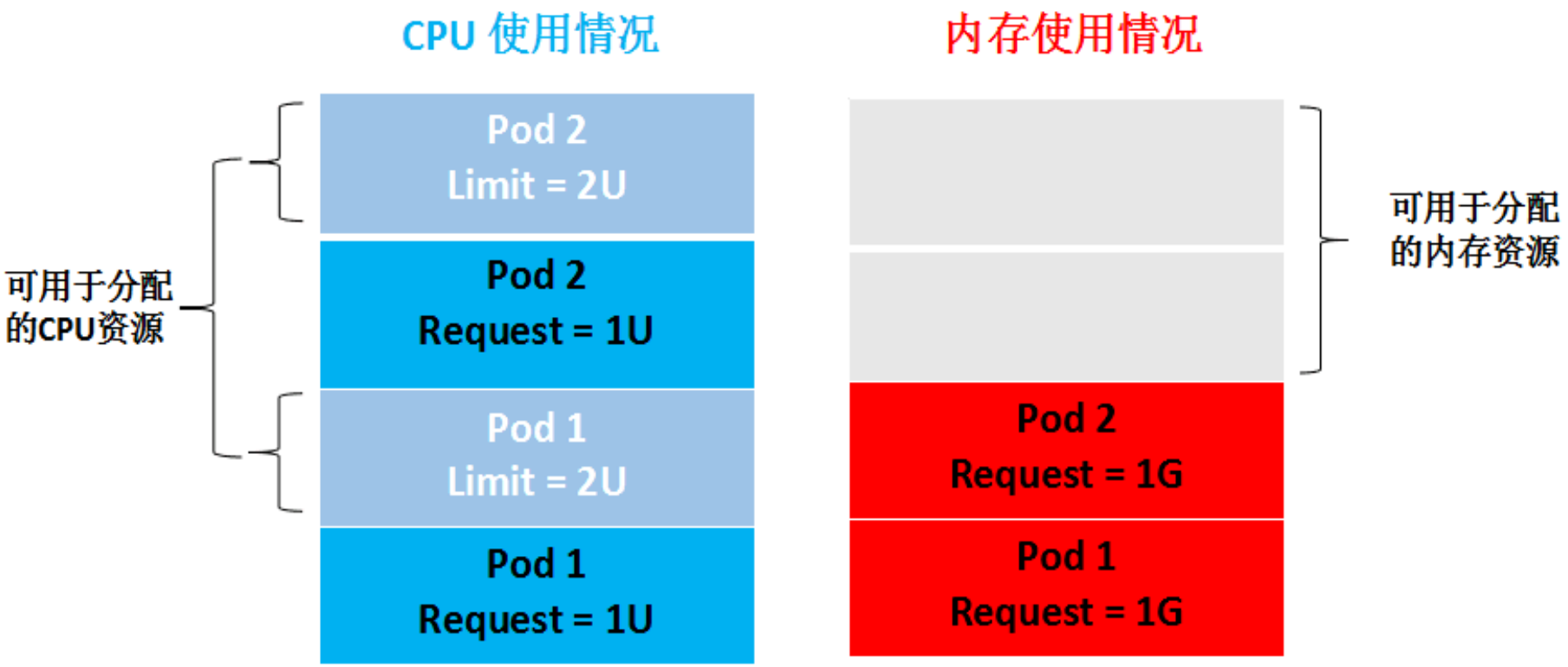
元素	规格	多维资源	使用率	下限	已用	可用	预测	上限	状态
实例	部分资源	全部资源维度	实例指标	基准性能	实际消耗	空闲	基于历史	规格	可调度、需调整、优先级、抢占
节点	机型	+机器属性	节点指标	预留	+实际消耗	+空闲	基于历史	规格	空闲、平衡、过载、能耗
集群	机型集合	+集群属性	聚合指标	备机	+实际消耗	+空闲	聚合预测	聚合规格	浪费、能耗、水位、库存、备机

1. 启发式
2. 多目标优化
3. 原生支持批量

Vector应用

- 调度
  - 节点打分，启发规则
  - 节点状态，资源不/紧张
  - 维度稀缺，不/可压缩
  - 原生支持超卖&抢占&优先级
- 评估
  - 集群状态（指标）
    - 资源浪费、闲置成本、平衡状态
    - 负载，与CPU利用率正相关

参考K8s： Request-Limit模型



（CPU Requst,CPU Limit,Memory Requst, Memory Limit）= (1U, 2U, 1G,1G)

原理分析

- 资源维度三元表达
  - Request (可调度)&Limit(后限制)&Usage(实际值)
- 资源空闲时
  - 节点上利用率低
  - 发挥Request=0的作用
- 资源紧张时，包括真/假紧张
  - 抢占，节点上的总资源 < 所有实例Limit的总和
    - 可压缩资源(CPU), 按照Request的比例分占CPU调度时间片
    - 不可压缩资源(Mem), 利用优先级高低驱逐(迁移)实例
      - 优先搞走Request=Limit=0
      - 其次搞走0<Request<Limit<Infinity
      - 保留0<Request=Limit, 除非节点剩余资源未达剩余资源的要求
- 极限情况，资源紧缺处理，真紧张
  - 不可压缩资源抢占的资源回收策略
  - SLA保证，突发预留

直接映射

RoadMap	Jdcloud	kubernetes	rms调度
现状 ① 所有CPU模式 ② 所有规格类型 ③ CpuAllocRatio	Flavor(cpu)	0 < Request(cpu) = Limit(cpu)	Bin packing
	VmAlloc(cpu)	Request(cpu) = Limit(cpu)	
	HostAvail(cpu)	HostAvail(cpu)	
基于利用率 ①②③ +MixMode	PredictVmUtil(cpu)	Request(cpu)	Bin packing
	Flavor(cpu)	Limit(cpu)	
	HostAvail(cpu)	HostAvail(cpu)	
基于Vector ①②③ +MixMode	PredictVmUtil(cpu)	Request(cpu)	Bin packing decreasing (BPD)
	Flavor(cpu)	Limit(cpu)	
	HostAvail(cpu)	HostAvail(cpu)	

复用策略

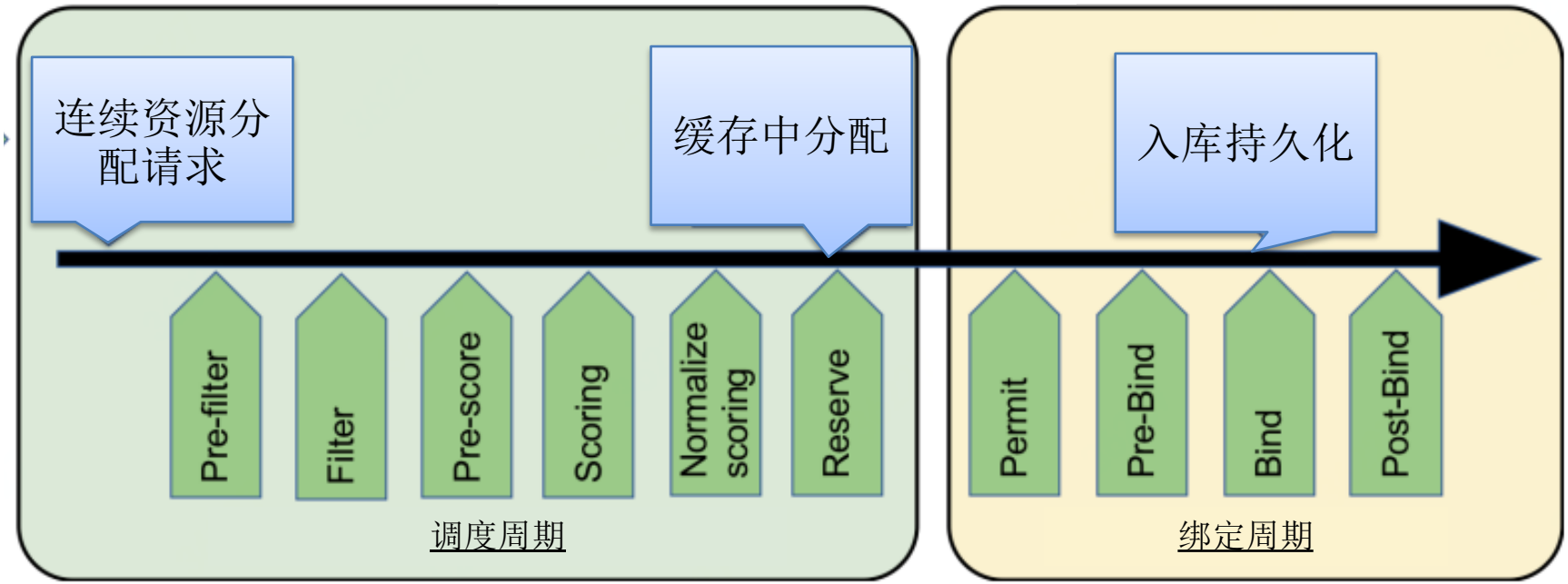
大类	小类	Context	控制	其他
实例	通用	一级调度|Request = Flavor	Limit = Flavor	
	抢占	一二级调度|Request = 0，离线业务	Limit = Flavor|基准	
	优先级	二级调度|Request + Limit	$\sum Limit > Host$	
	超卖	一二级调度|Request = [0, 预测Limit]		
节点	绑核	实际扣减，在线业务		
	超卖	等价扣减，弹性+通用业务		
	碎片	超卖 + Consolidation		
	功耗	基于利用率模型	$\min \bigcup$(节点 + 机架)	

架构目标

- Scheduler Framework
  - 轻量化, Framework Core@RMS
  - 组件化; 接口化,
  - 内置默认调度
- 所有策略插件化
  - 抢占&预占&突发
  - 可配置
- 标签计算

框架

分配调度上下文



栏目	备注
时间窗口	两个周期：调度周期（充分利用缓存数据）、绑定周期（利用状态数据、持久化）
阶段	预选+优选+绑定
Qos	Request + Limit (涉及驱逐/GC策略，节点超卖策略，)

量化目标

- 装箱节点数
  - 算法，约束条件$BFD(I) \leq \frac{11}{9} OPT(I) + 1$
  - 标签合并，调度空间
  - Incremental阶段
- 碎片化，资源浪费率WASTE
  - 节点利用率分类，状态转移，代价
  - Consolidation阶段
- 能耗，资源利用和功耗相关性
  - 空机SLEEP|WAKEUP
  - 采购优化
- 售卖率|容量管理|库存服务|经济模型
  - 运营决策，套餐

功耗

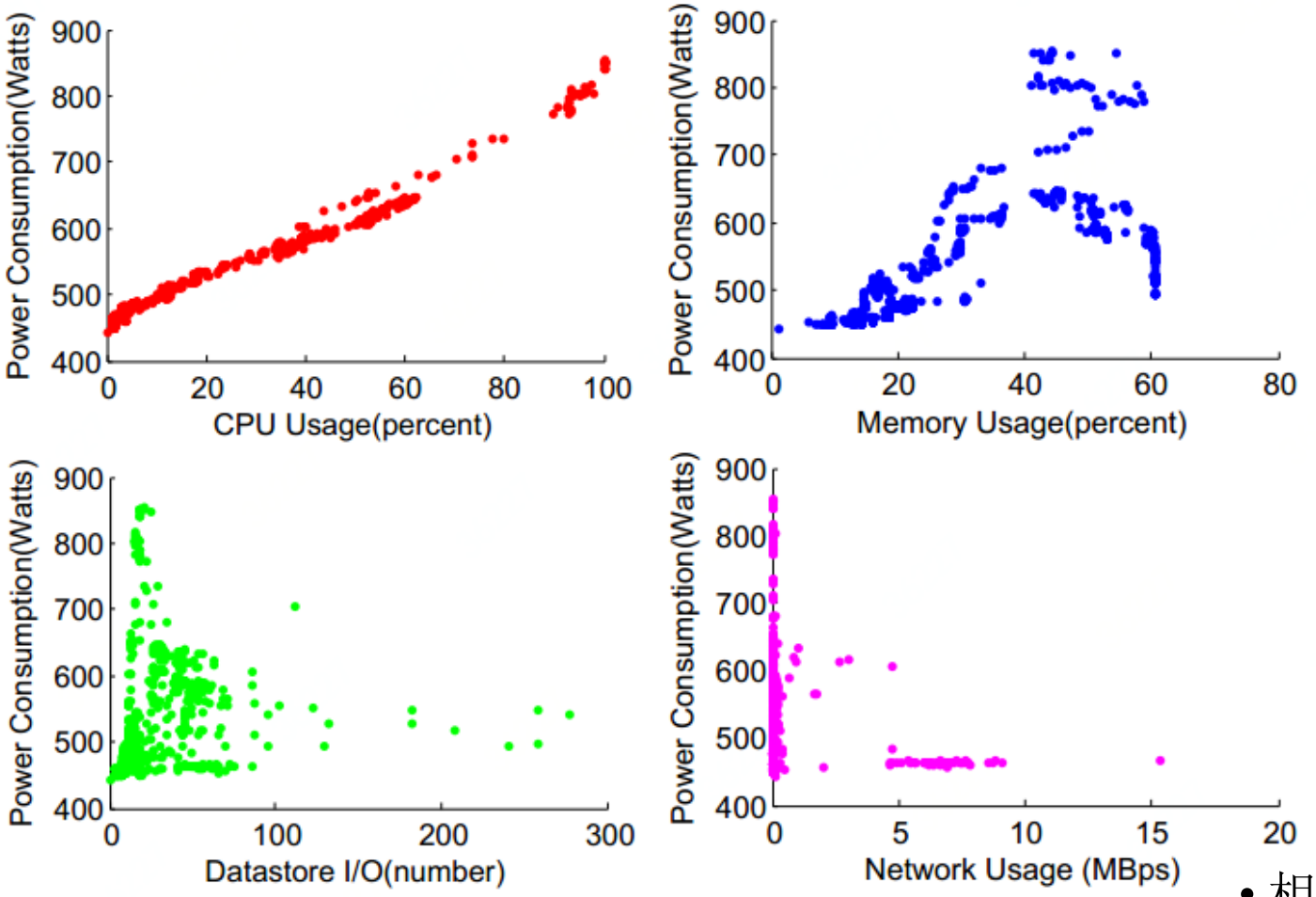


Fig. 3: Relationship between resource usages and power consumption

Table 1: VM instances in the data center

Variable a	Variable b	Correlation Coefficient(a,b)
Power Consumption	CPU Usage	0.990667
	Memory Usage	0.681875
	Datastore I/O	0.253151
	Network Usage	-0.12116

- 相关性结论[1]
  - CPU与能耗高度正相关
  - 网络与能耗无关
  - 磁盘，内存相关性偏弱
- 资源利用和能耗
  - 相关因子

功耗-约束

- 机架因素
- 供电额度
- GPU
- 四代机
- TOR拓扑

相关性结论

- Idle机器耗能 = 70% * Full机器耗能
  - Sleep状态
- 能耗(cpu利用率)：
  - 功率： $P(u) = k \cdot P_{max} + (1-k) \cdot P_{max} \cdot u$
  - 能耗： $E = \int_{t_0}^{t_1} P\big(u(t)\big)dt$
- 节点&实例Cpu利用率
  - Share, Set和Mix是统一的
  - 规格系数为权重
  - 绑核和时间片计量上是统一的
- **所有量化都统一到利用率及其预测上**

可观测目标

- 可扩展
  - 架构
- 可度量
  - **量化目标**中的衍生指标
- 可评估
  - 包含衍生指标的统计分析
- 其他
  - 无侵入性，灵活

实现

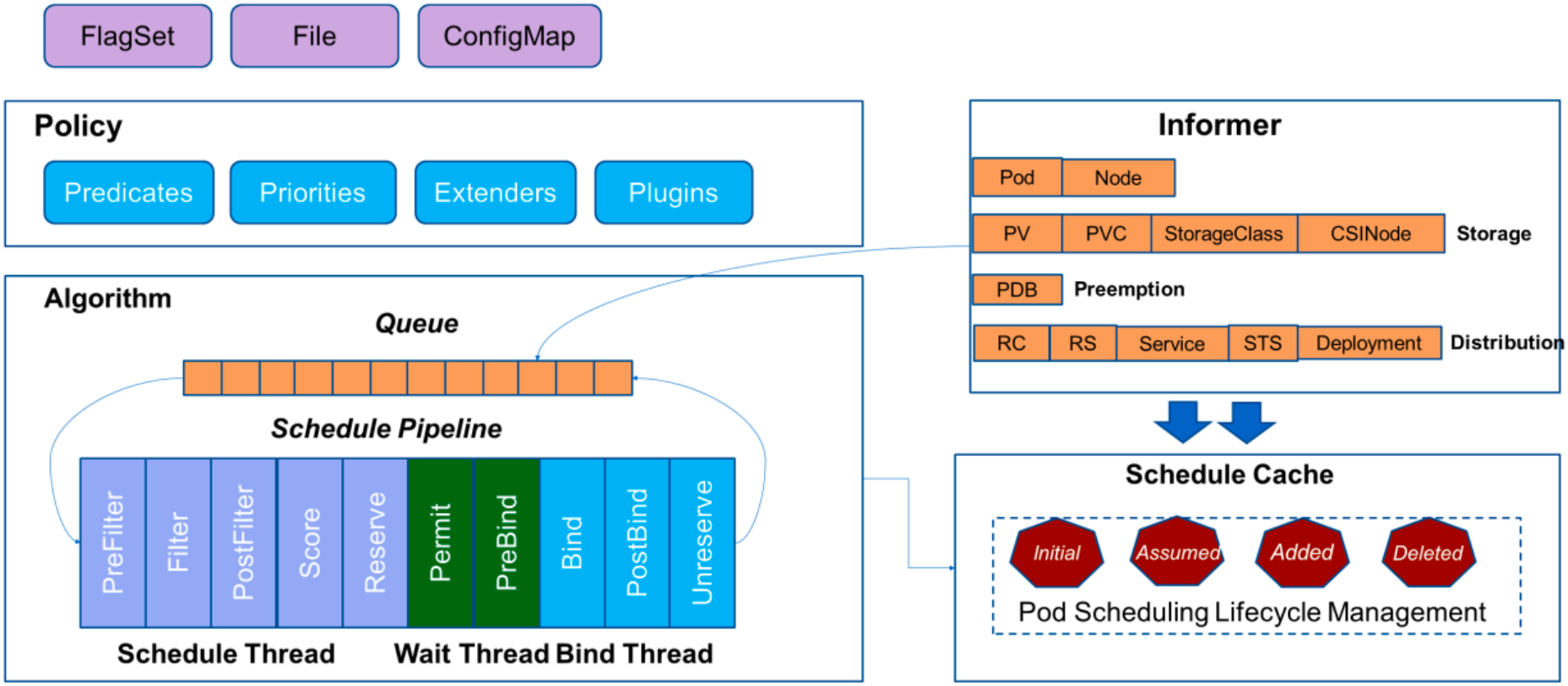
- Scheduler Framework
  - 提供插件化和默认调度（兼容）
- 多目标优化
  - Best effort
    - 最大化利用率
    - 最小化浪费
    - 最小化能耗
  - Tradeoff
    - 效能提升优先，一级调度
    - 隔离性兼顾，二级调度

参考

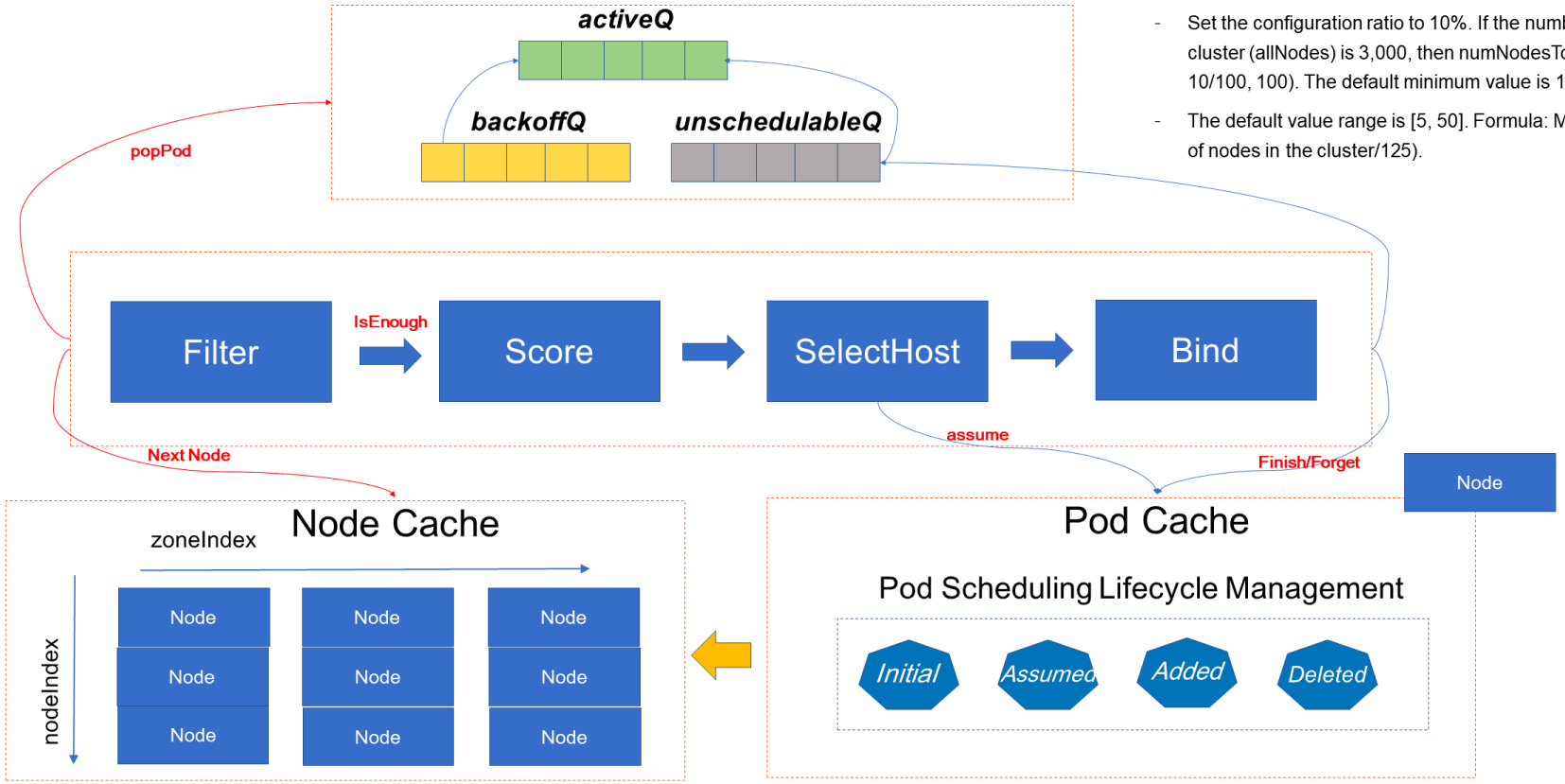
- [1] Virtual machine consolidated placement based on multi-objective biogeography-based optimization

Backup

- Kubernetes Scheduler


FlagSet
File
ConfigMap
Policy
Predicates
Priorities
Extenders
Plugins
Informer
Pod
Node
PV
PVC
StorageClass
CSINode
Storage
PDB
Preemption
RC
RS
Service
STS
Deployment
Distribution
Algorithm
Queue
Schedule Pipeline
PreFilter
Filter
PostFilter
Score
Reserve
Permit
PreBind
Bind
PostBind
Unreserve
Schedule Thread
Wait Thread
Bind Thread
Schedule Cache
Initial
Assumed
Added
Deleted
Pod Scheduling Lifecycle Management

Scheduling Process



Configure the node sampling scope

- Set the configuration ratio to 10%. If the number of nodes in the cluster (allNodes) is 3,000, then numNodesToFind = Max(3000 x 10/100, 100). The default minimum value is 100.
- The default value range is [5, 50]. Formula: Max(5, 50 - Number of nodes in the cluster/125).

- LeastRequestedPriority: prioritizes distribution.
- MostRequestedPriority: prioritizes stacking.
- BalancedResourceAllocation: indicates the fragmentation rate.
- RequestedToCapacityRatioPriority: indicates the score of the ratio.

- Concepts used in the resource usage formula
  - Request: resources that have been allocated to nodes
  - Allocatable: resources that can be scheduled to nodes.
- Score of preferential distribution
  - (Allocatable - Request) / Allocatable * Score
- Formula for calculating preferential stacking
  - Request / Allocatable * Score
- Fragmentation rate
  - {1 - Abs[CPU(Request/Allocatable) - Mem(Request/Allocatable)]}* Score
- Additional information:
  - Based on the preceding scenarios, a score of 0 is given in all scenarios where Request >= Capacity.
- Specified ratio
  - MapConfig of a score with the specified ratio (0:x, 1:x, 2:x, …, 99:x , 100:x)
  - Formula：Request / Allocatable * MaxUtilization

Priorities – Pod Distribution

- Solved problems
  - The pod distribution feature allows you to spread matched pods across different topologies for deployment.
- SelectorSpreadPriority
  - Collect statistics on nodes.
  - TopoPods = Selector condition for the workload of the controller with Exists Pod matching Income Pod
  - (Sum(TopoPods) – TopoPods)/Sum(TopoPods)
- ServiceSpreadingPriority
  - An official source indicates that ServiceSpreadingPriority is very likely to replace SelectorSpreadPriority.
  - Collect statistics on nodes.
  - TopoPods: compliant with the selector condition specified by the service where pods are located
  - (Sum(TopoPods) – TopoPods)/Sum(TopoPods)
- EvenPodsSpreadPriority
  - topologyKey indicated by spec
  - TopoPods = spec-compliant labelSelector
  - Score calculation formula
    - NodeTopoPods = cumulative TopoPods at the node level
    - MaxDif = Max(NodeTopoPods) – Min(NodeTopoPods)
    - NodeTopoPods - MaxDif/MaxDif

- NodeAffinityPriority
  - TopoPods: cumulative instances of affinity compliance
  - TopoPods/Max(TopoPods) x MaxPriority
- ServiceAntiAffinity
  - Evenly distribute the pods in a service based on a label value of the node.
- NodeLabelPrioritizer
  - Preferentially distribute pods to a node with or without a specific label.
- ImageLocalityPriority:
  - Perform scheduling based on image affinity.

Configure the Kubernetes Scheduler

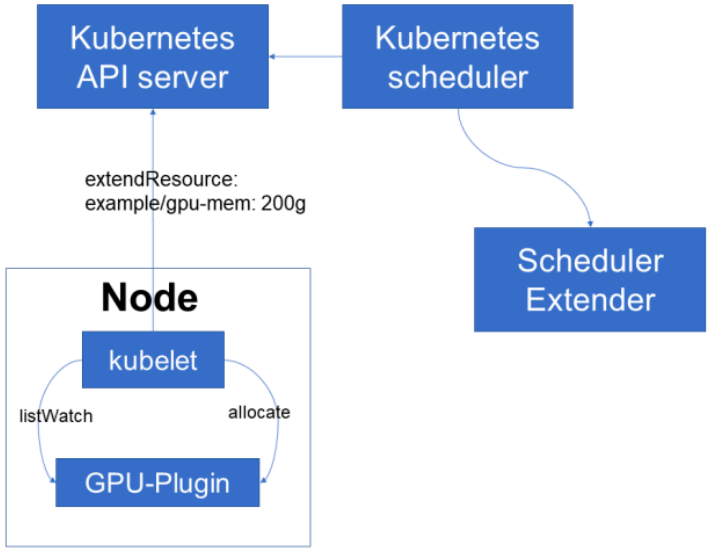
- Start the Kubernetes scheduler
  - Default configuration (--write-config-to)
  - Configuration file (--config)
- Configuration file description
  - schedulerName: scheduling based on Pod.SchedulerName
  - algorithmSource: used to configure algorithms
  - hardPodAffinitySymmetricWeight: used to configure the affinity weight
  - percentageOfNodesToScore: the threshold for exit when the ratio of the filtered nodes to all nodes is equal to this value
  - bindTimeoutSeconds: the timeout period of the binding phase

```
# Startup with the default configuration
kube-scheduler \
    --kubeconfig=/etc/kubernetes/kubeconfig/scheduler.kubeconfig \
    --v=3
# Startup with the specified configuration file, which is recommended when custom
  parameters are required.
kube-scheduler \
    --config=/home/admin/scheduler/conf/scheduler-plicy.config \
    --v=3
```

Scheduler Extender

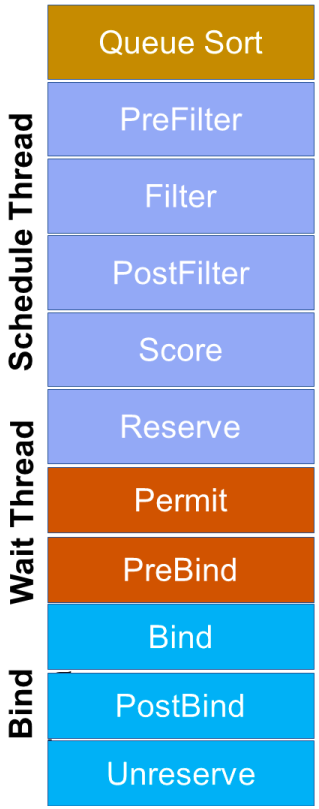
- What can we do?
  - Keep the original scheduling code and directly start the service as a plugin to be called by the Kubernetes scheduler (similar to webhook).
  - Support injection of predicate, preempt, priority, and bind.
  - A type of extended resource, which can be bound to only one extender.
- Configuration procedure
  - Configuration file description
- Example
  - Apply for GPU memory. Only the extender knows the memory size of each GPU, so add the extender filter.

```yaml
kind: Policy
apiVersion: v1
extenders:
- urlPrefix: "http://xxxx:xxxx/scheduler-gpu-extender"
  filterVerb: filter
  weight: 1
  enableHttps: false
  # 调度器传递nodenames列表，而不是nodeinfo 列表
  nodeCacheCapable: true
  # 调用extender服务报错或者网络不可达的时候，是否可忽略extender
  ignorable: false
  managedResources:
  - name: "example/gpu-mem"
  # resourceFit阶段是否忽略这个资源的校验
    ignoredByScheduler: false
```



Scheduler Framework

- Extension point usage
- Concurrency model

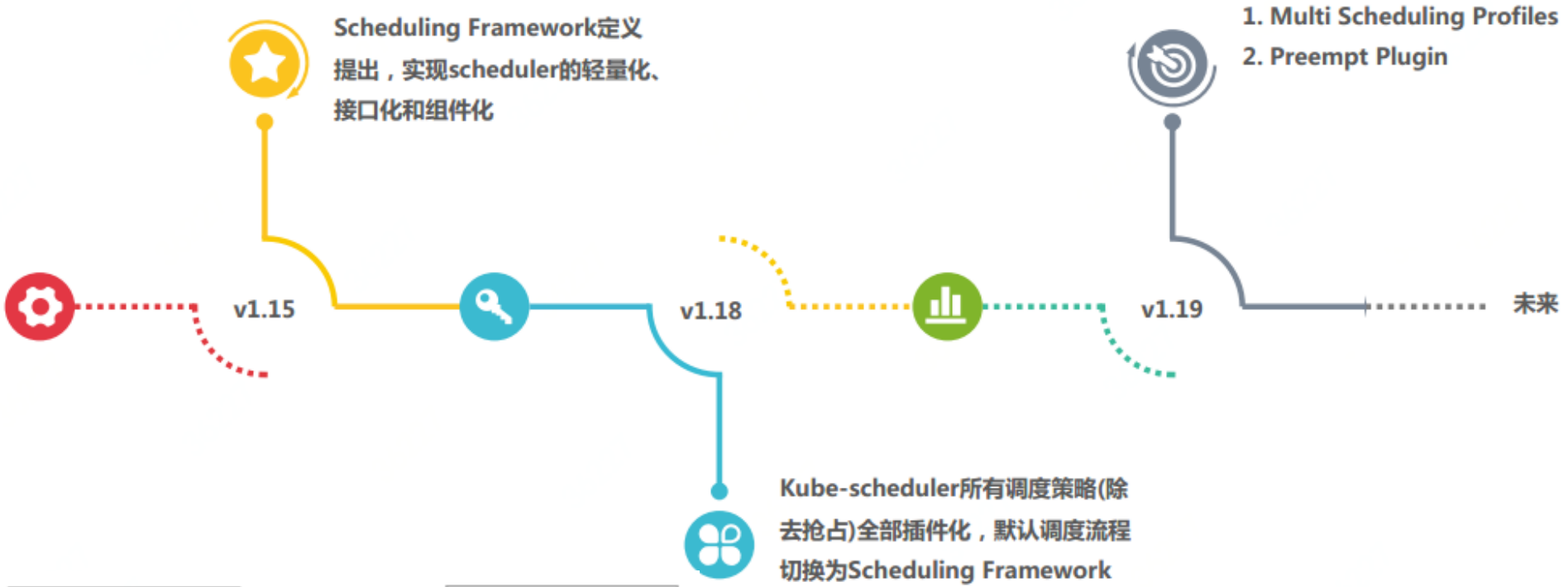


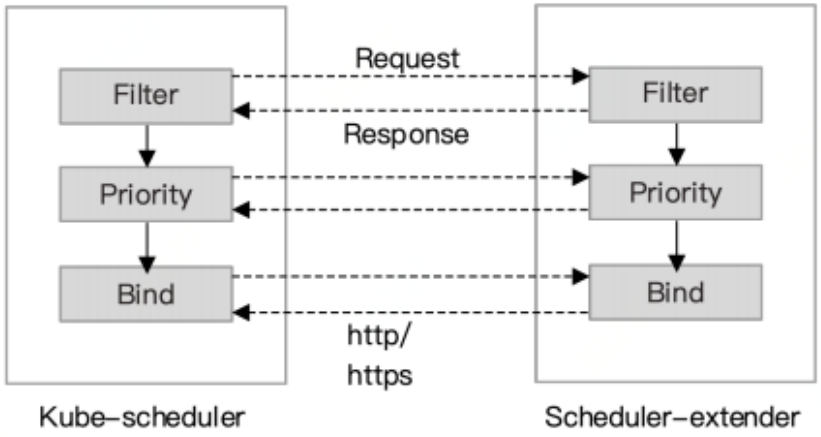
- QueueSort: supports the sorting of custom pods.
- PreFilter: preprocesses pod requests.
- Filter: creates custom filter logic.
- PostFilter: processes logs and metrics or preprocesses data before the score phase.
- Score: creates custom score logic.
- Reserve: records the memory usage of resources through stateful plugins.
- Permit: supports the Wait, Deny, and Approve actions, which can be used as Gang insertion points.
- PreBind: performs operations, such as attaching disks to nodes, before nodes are bound.
- Bind: processes a single pod through a single BindPlugin.
- PostBind: implements the logic after successful binding.
- Unreserve: implements rollback when an error is returned in any phase from Permit to Bind.

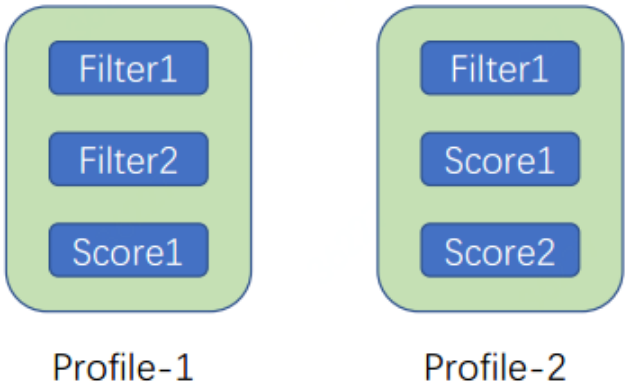
问题

- Label 划分资源池
  - 不利于资源利用率
- 调度性能
  - Qps
  - 成功率 (race condition)
  - 批量 (dp)

现状


Scheduling Framework定义
提出，实现scheduler的轻量化、
接口化和组件化
1. Multi Scheduling Profiles
2. Preempt Plugin
v1.15
v1.18
v1.19
未来
Kube-scheduler所有调度策略(除
去抢占)全部插件化，默认调度流程
切换为Scheduling Framework

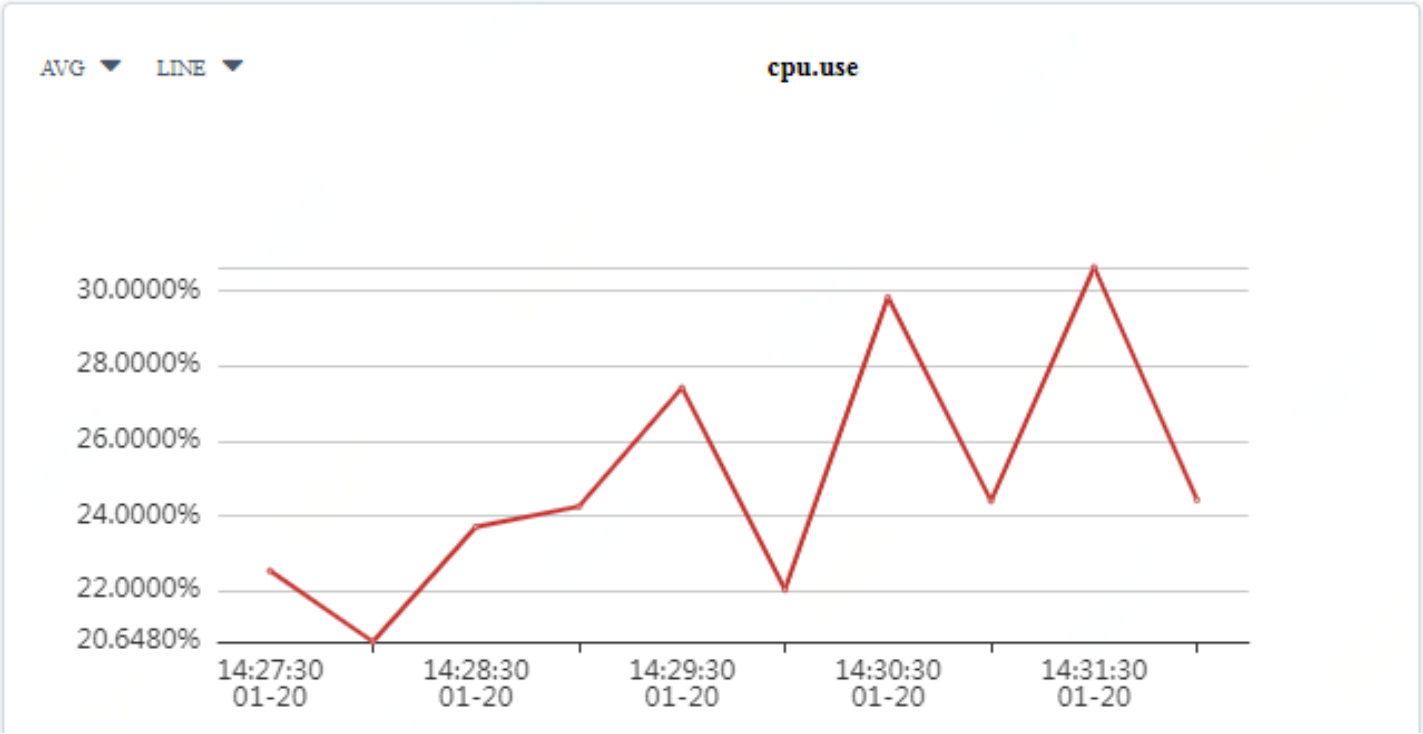

Filter
Priority
Bind
Request
Response
http/
https
Filter
Priority
Bind
Kube-scheduler
Scheduler-extender


Filter1
Filter2
Score1
Filter1
Score1
Score2
Profile-1
Profile-2

Our现状

节点&实例

- 10.217.35.88
- 2021-01-20 14:27:43 - 2021-01-20 14:32:43



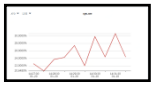
i-zg7l9bs3mf
i-7n345sywim
i-uqmi1k87gq
i-fx4mvsycow
i-09mnv9uiju
i-57bfupdns0
i-77dqm7dyph
i-r3zxvcb95z
i-j8mr1w2tqm
i-rul8xuniqh
i-e180y4gl3s
i-iwa93w60s3
i-18c109qsx9
i-0wt7gkumiq
i-d2aytdu16k
i-egwt92d5ok
i-rtp6knj9np
i-6dtdntq4sl
i-b3oc3w8dlc
i-fukjbbow8d
i-9vfligkvxs
i-hzudsnlhn6
i-gmjzhe1cyi
i-vtqb6skyf3
i-vhqbz9h8r8
i-xb1pqdqguw
i-910ju1jhdd

物理机：10.217.35.88

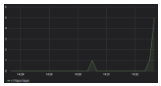
时间段：2021-01-20 14:27:43 - 2021-01-20 14:32:43

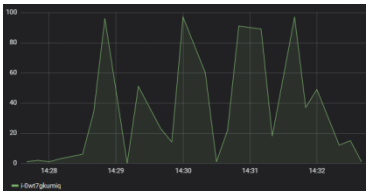
资　源：已全部售卖

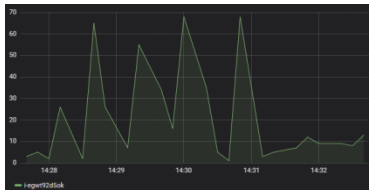
物理机：

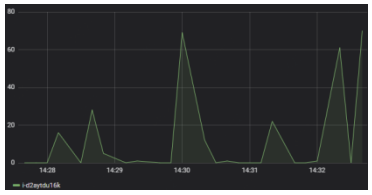


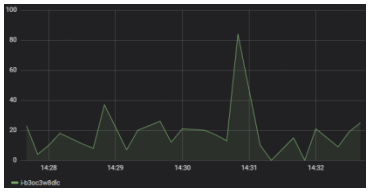
实例：

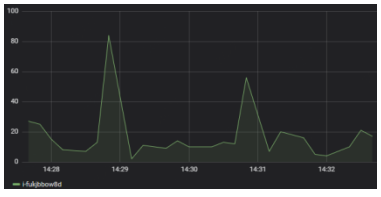


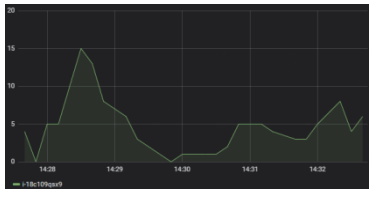


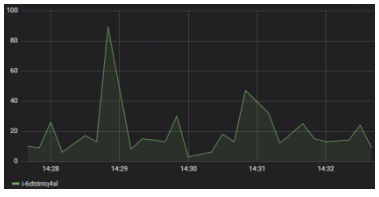


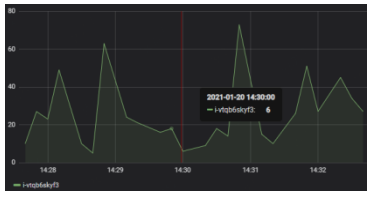


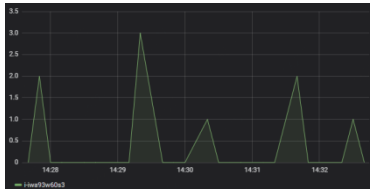


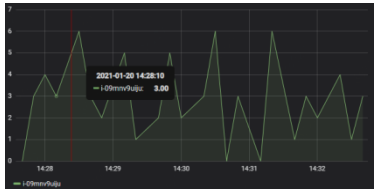


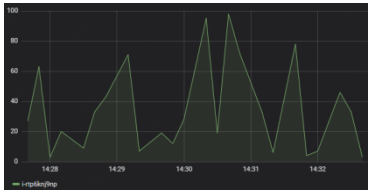


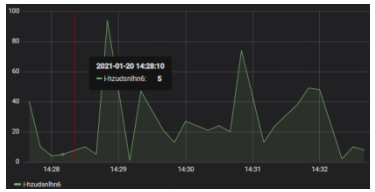


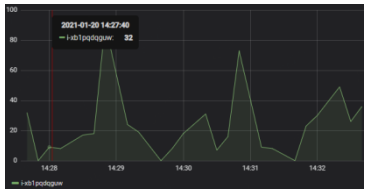


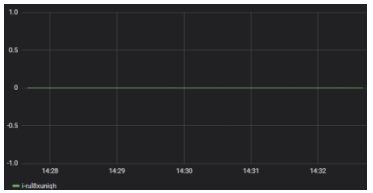


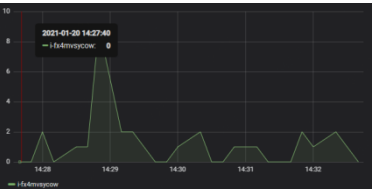


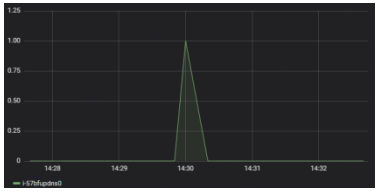


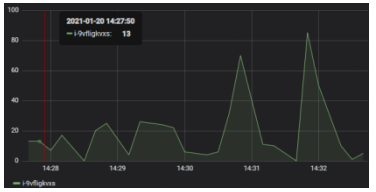


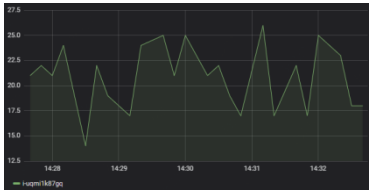






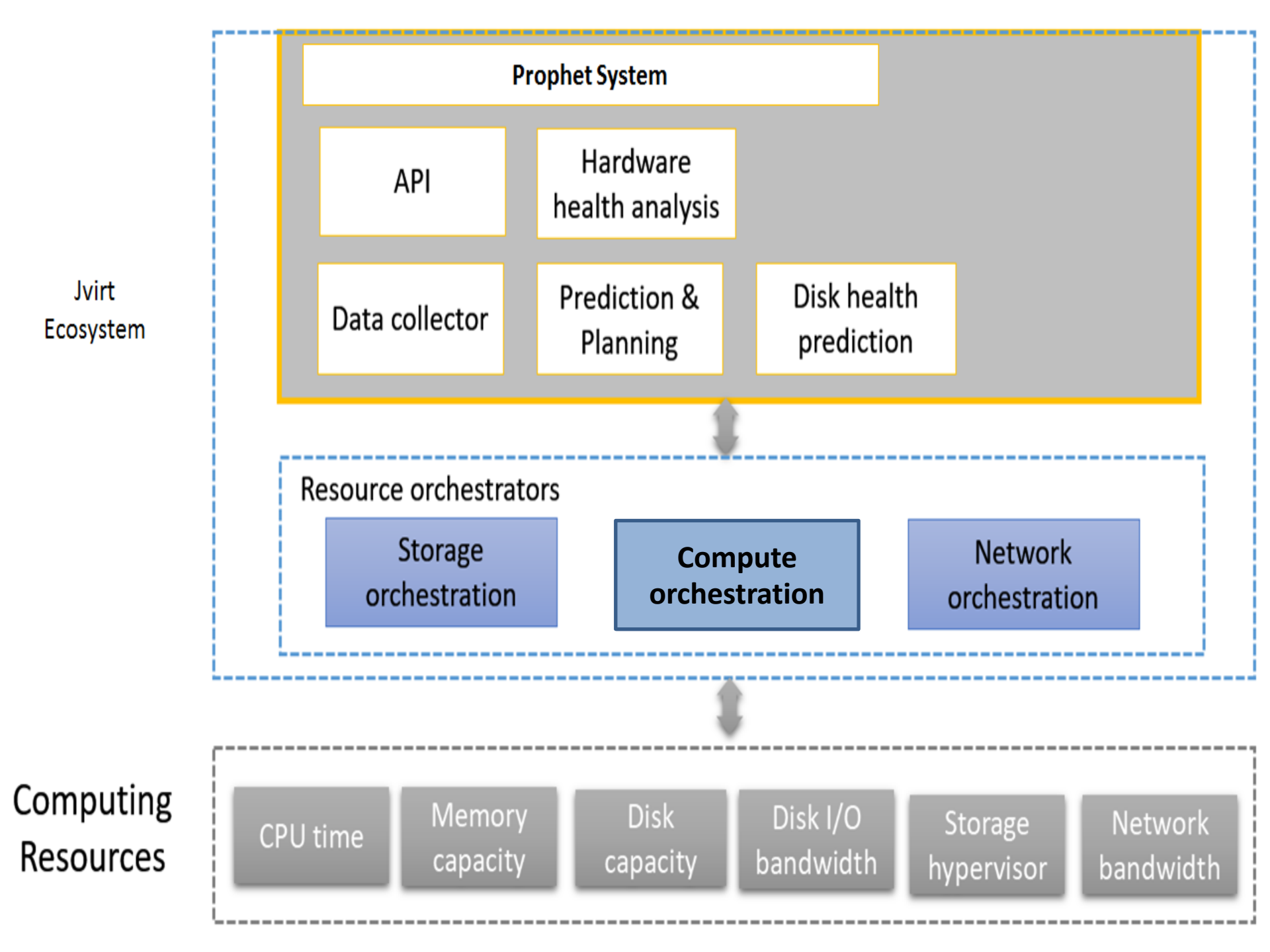


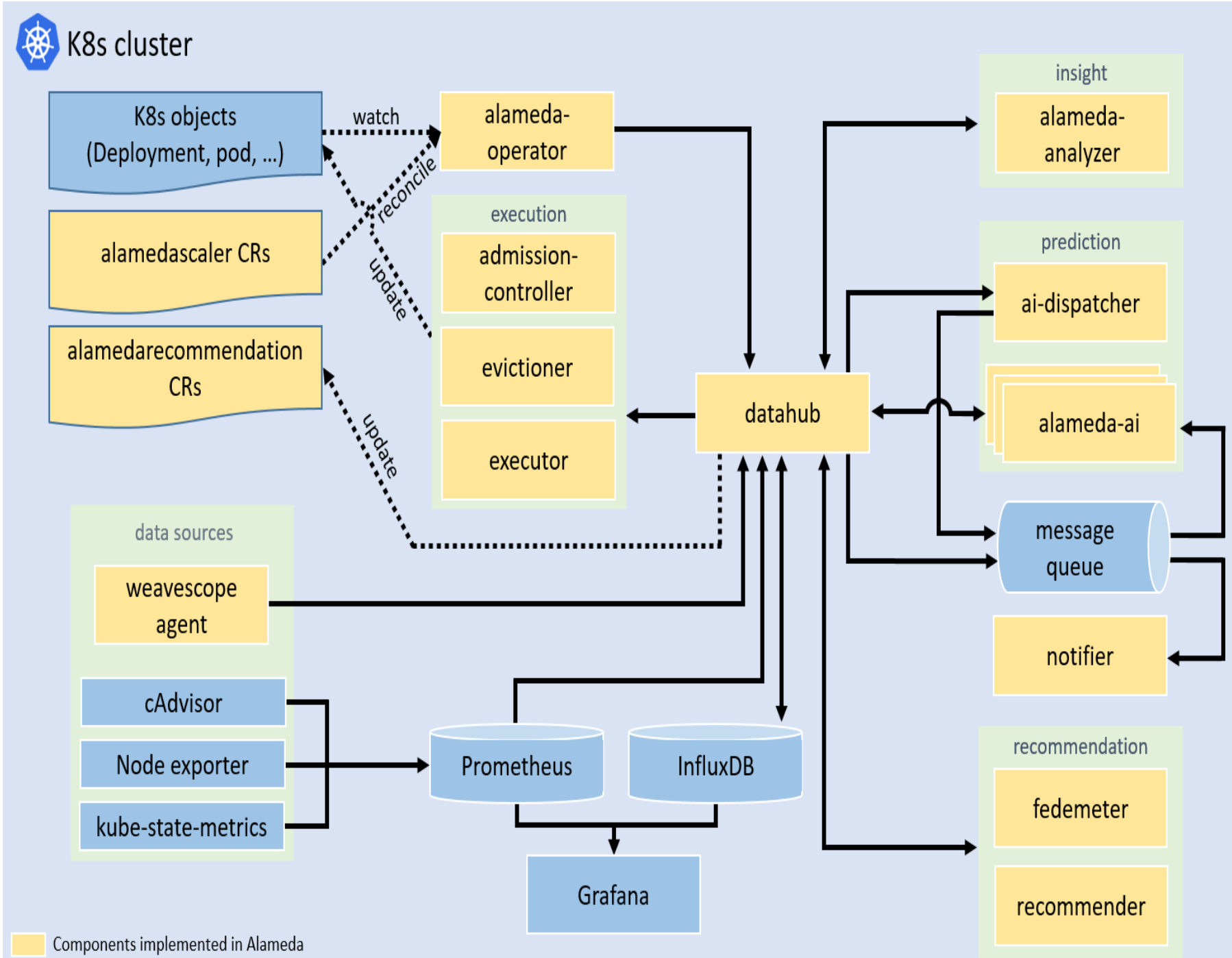


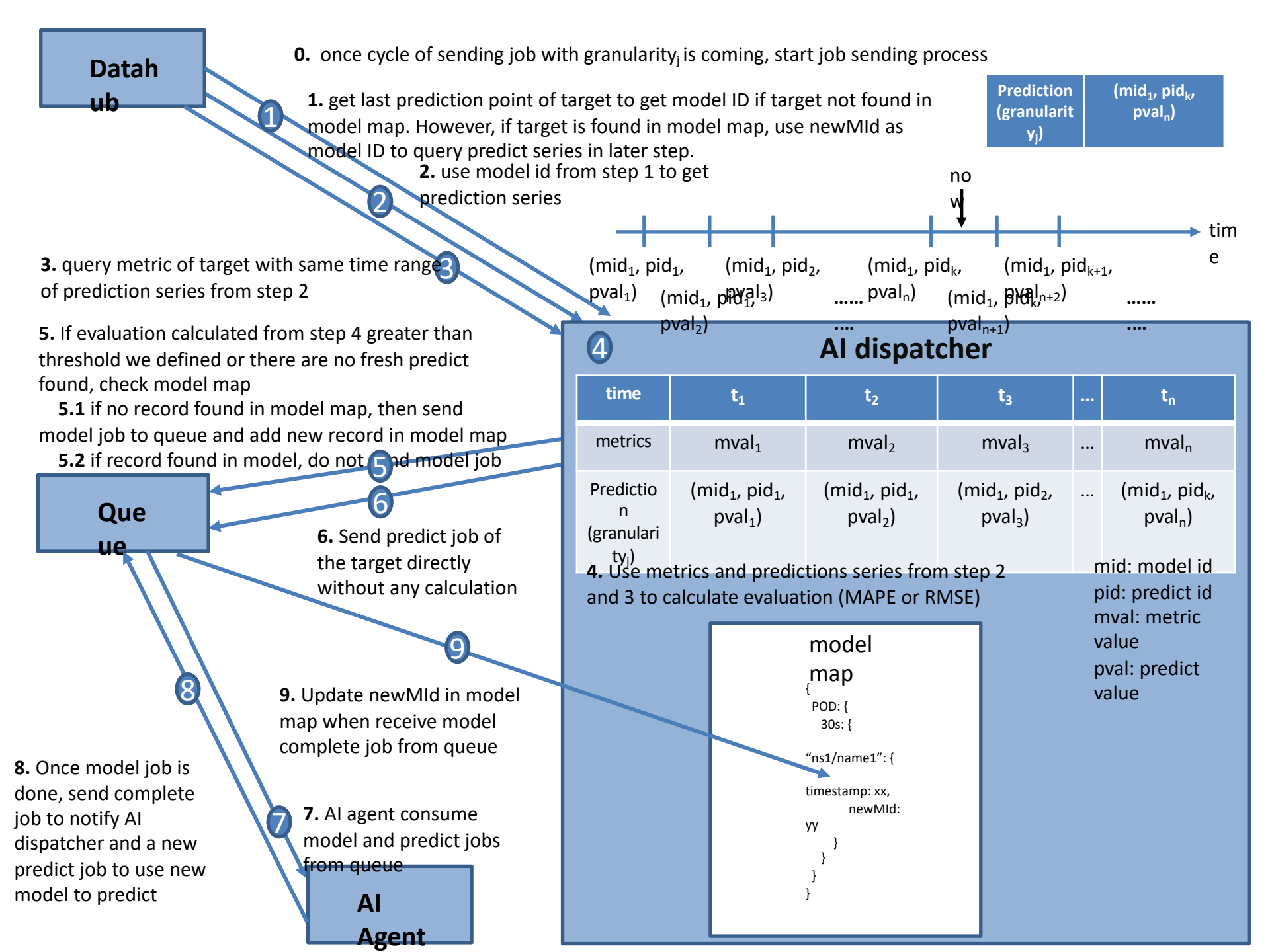


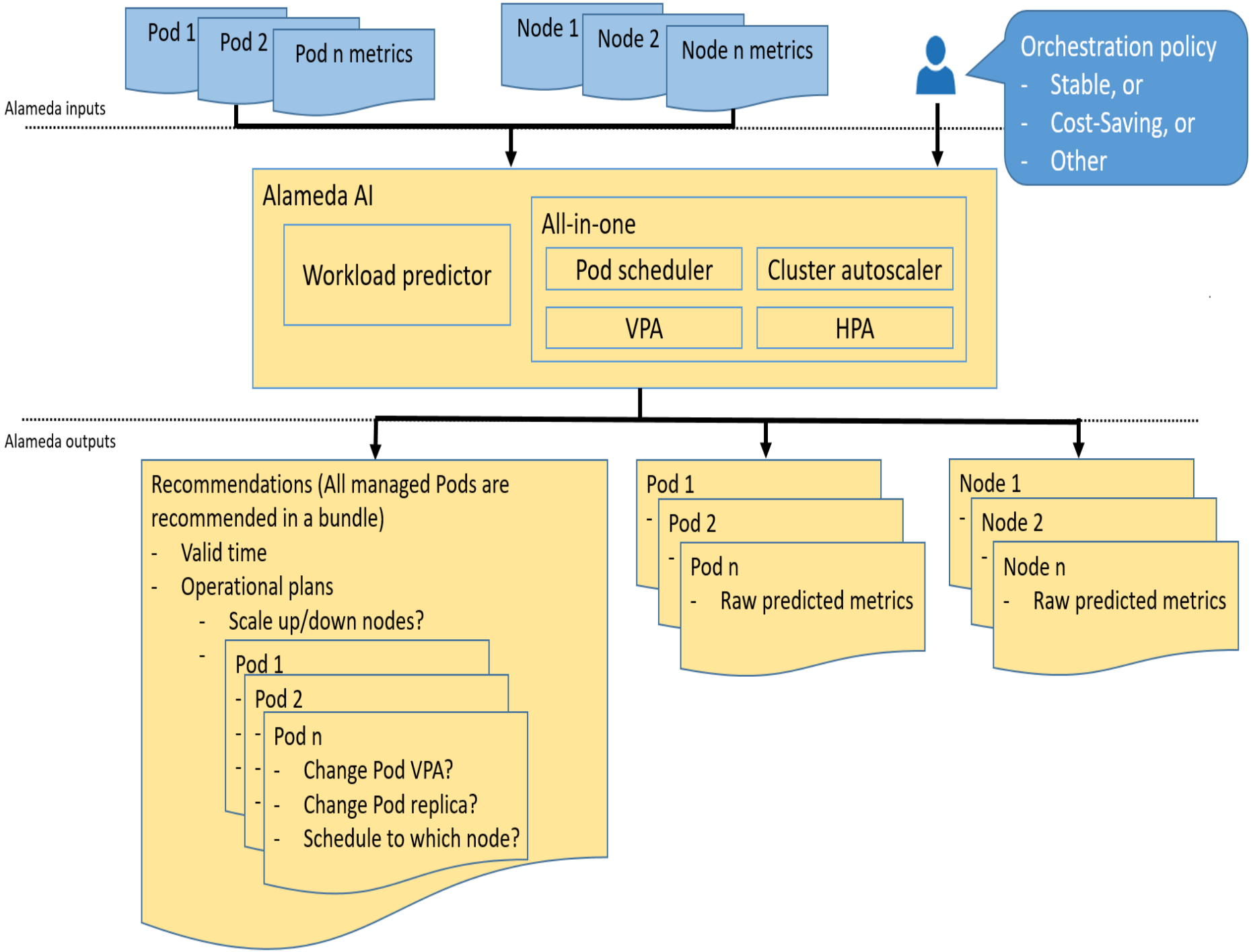
Proposal

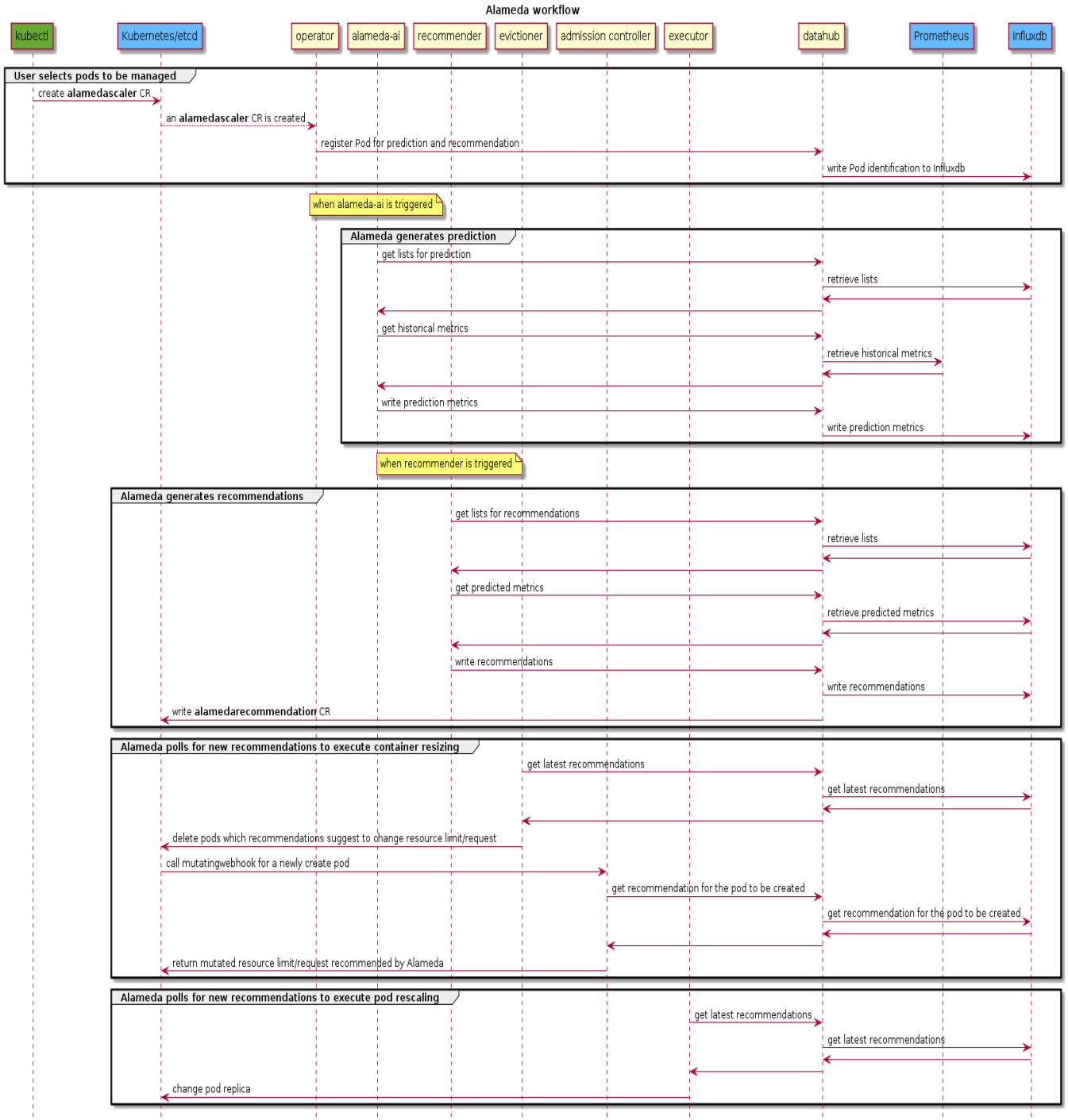
- Xxxx is a prediction engine that foresees future resource usage of cluster from the cloud layer down to the instance level. We use machine learning technology to provide intelligence that enables dynamic scaling and scheduling of your instances - effectively making us the “brain” of cluster resource orchestration. By providing full foresight of resource availability, demand, health, impact and SLA, we enable cloud strategies that involve changing provisioned resources in real time.

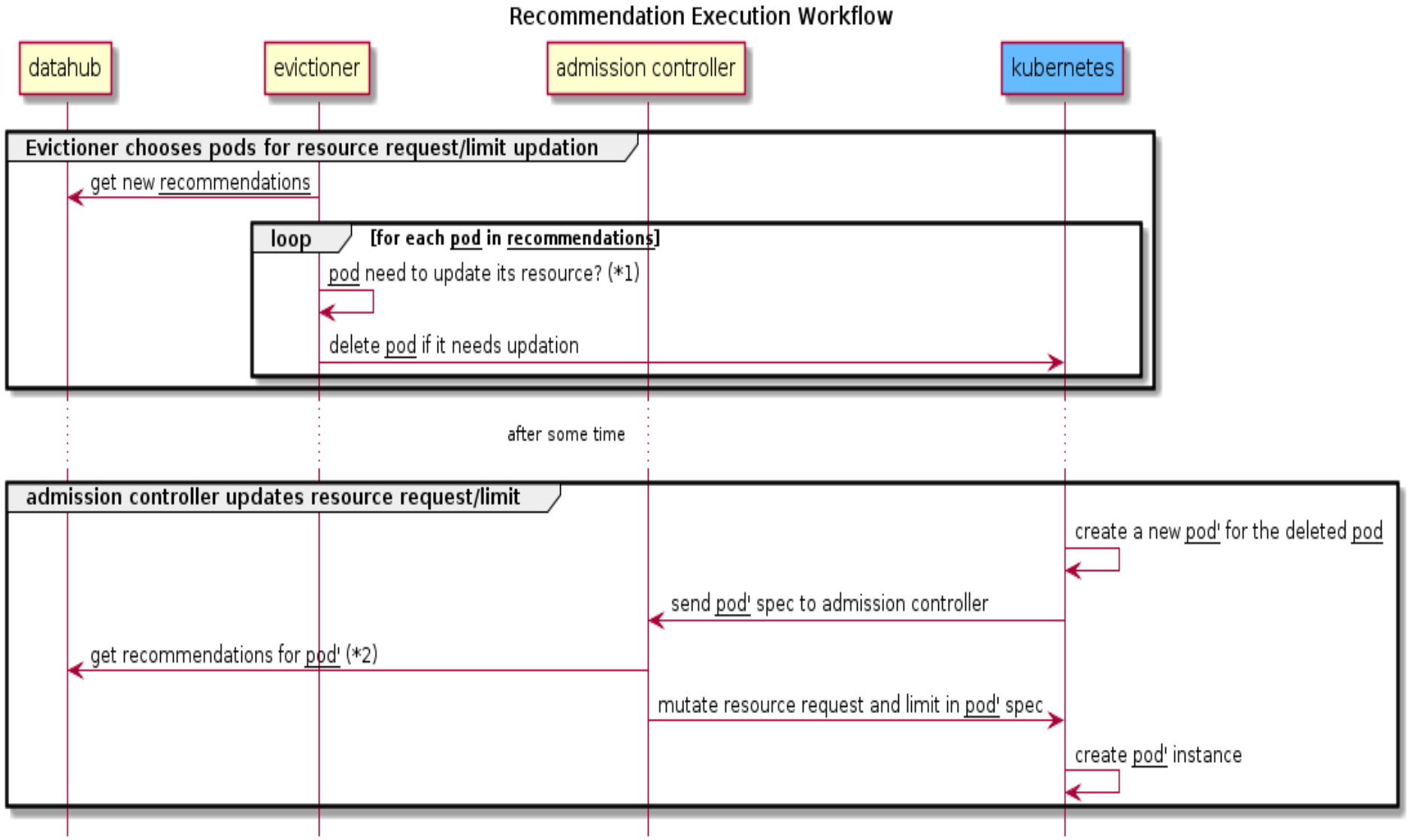

Jvirt
Ecosystem
Prophet System
API
Hardware
health analysis
Data collector
Prediction &
Planning
Disk health
prediction
Resource orchestrators
Storage
orchestration
Compute
orchestration
Network
orchestration
Computing
Resources
CPU time
Memory
capacity
Disk
capacity
Disk I/O
bandwidth
Storage
hypervisor
Network
bandwidth

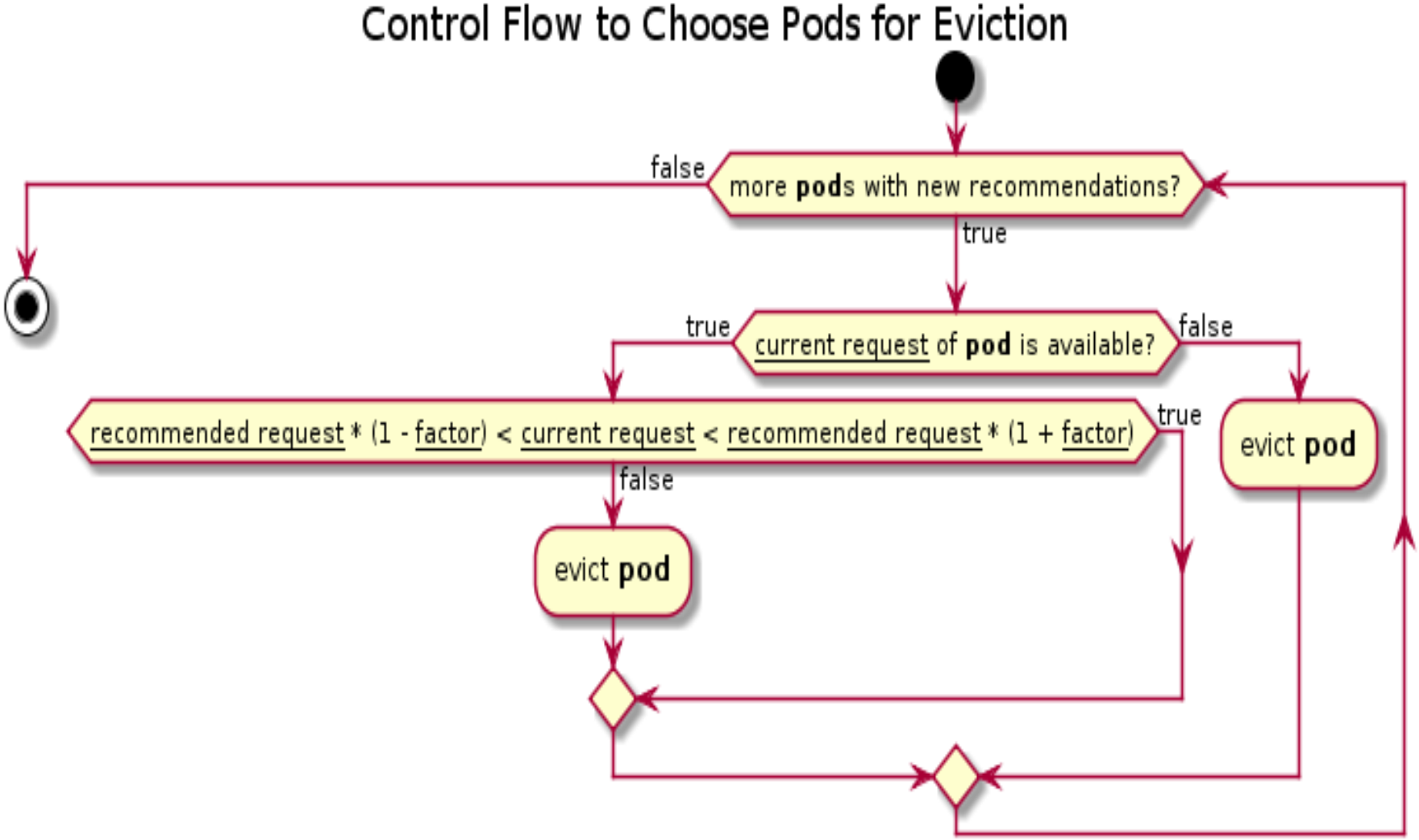

K8s cluster
K8s objects (Deployment, pod, ...)
watch
alameda-operator
reconcile
update
alamedascaler CRs
alamedarecommendation CRs
update
execution
admission-controller
evictioner
executor
datahub
insight
alameda-analyzer
prediction
ai-dispatcher
alameda-ai
message queue
notifier
data sources
weavescope agent
cAdvisor
Node exporter
kube-state-metrics
Prometheus
InfluxDB
Grafana
recommendation
fedemeter
recommender
Components implemented in Alameda

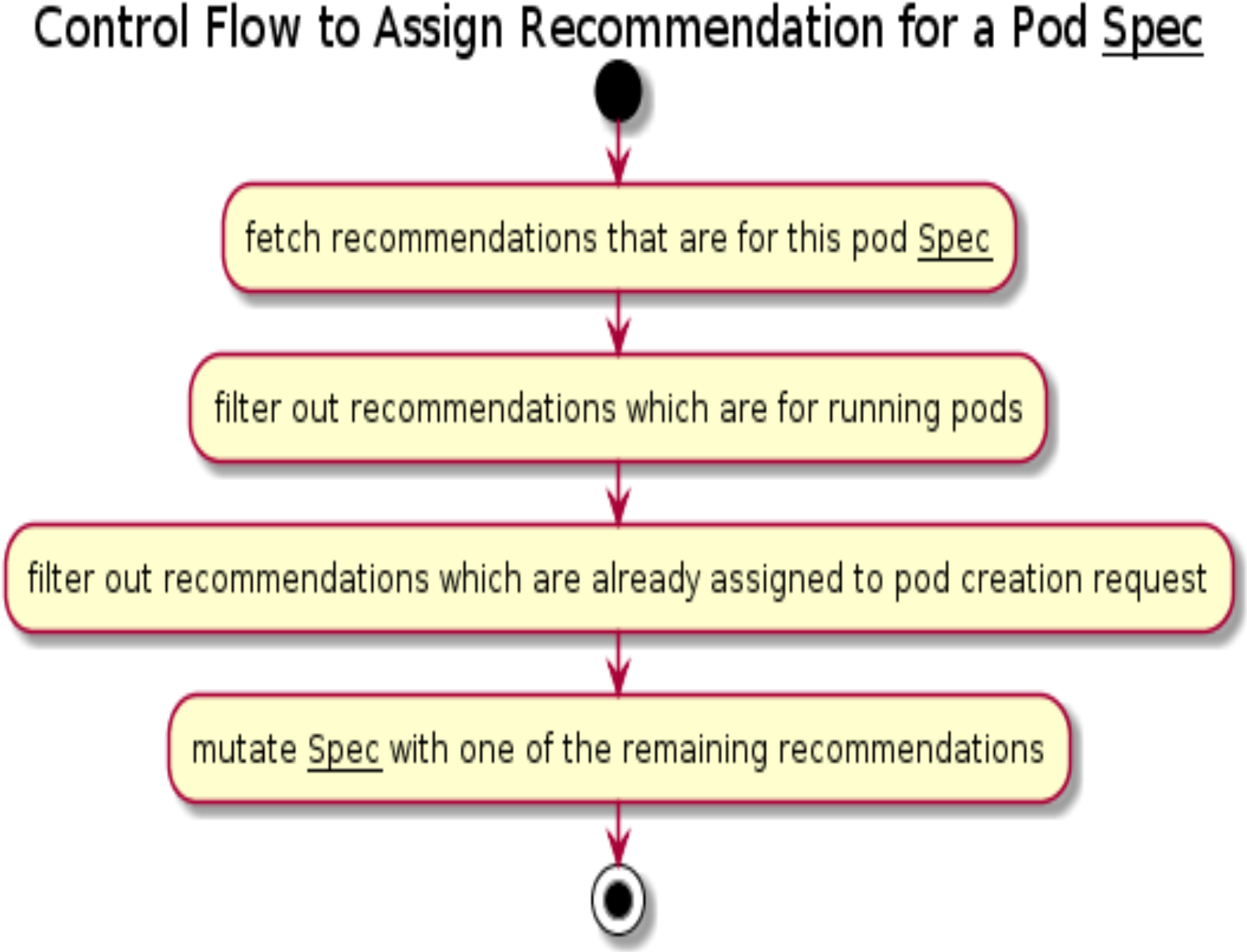

Datahub
0. once cycle of sending job with granularity$_j$ is coming, start job sending process
1. get last prediction point of target to get model ID if target not found in model map. However, if target is found in model map, use newMId as model ID to query predict series in later step.
2. use model id from step 1 to get prediction series
Prediction (granularity$_j$)
(mid$_1$, pid$_k$, pval$_n$)
now
time
(mid$_1$, pid$_1$, pval$_1$)
(mid$_1$, pid$_1$, pval$_2$)
(mid$_1$, pid$_2$, pval$_3$)
....
(mid$_1$, pid$_k$, pval$_n$)
(mid$_1$, pid$_k$, pval$_{n+1}$)
(mid$_1$, pid$_{k+1}$, pval$_{n+2}$)
....
3. query metric of target with same time range of prediction series from step 2
4
AI dispatcher
time | t$_1$ | t$_2$ | t$_3$ | ... | t$_n$
metrics | mval$_1$ | mval$_2$ | mval$_3$ | ... | mval$_n$
Prediction (granularity$_j$) | (mid$_1$, pid$_1$, pval$_1$) | (mid$_1$, pid$_1$, pval$_2$) | (mid$_1$, pid$_2$, pval$_3$) | ... | (mid$_1$, pid$_k$, pval$_n$)
4. Use metrics and predictions series from step 2 and 3 to calculate evaluation (MAPE or RMSE)
mid: model id
pid: predict id
mval: metric value
pval: predict value
5. If evaluation calculated from step 4 greater than threshold we defined or there are no fresh predict found, check model map
5.1 if no record found in model map, then send model job to queue and add new record in model map
5.2 if record found in model, do not send model job
Queue
6. Send predict job of the target directly without any calculation
model map
{
POD: {
30s: {
“ns1/name1”: {
timestamp: xx,
newMId: yy
}
}
}
}
9. Update newMId in model map when receive model complete job from queue
8. Once model job is done, send complete job to notify AI dispatcher and a new predict job to use new model to predict
7. AI agent consume model and predict jobs from queue
AI Agent


Pod 1
Pod 2
Pod n metrics
Node 1
Node 2
Node n metrics
Orchestration policy
- Stable, or
- Cost-Saving, or
- Other
Alameda inputs
Alameda AI
Workload predictor
All-in-one
Pod scheduler
Cluster autoscaler
VPA
HPA
Alameda outputs
Recommendations (All managed Pods are recommended in a bundle)
- Valid time
- Operational plans
- Scale up/down nodes?
-
Pod 1
- Pod 2
- Pod n
- Change Pod VPA?
- Change Pod replica?
- Schedule to which node?
Pod 1
- Pod 2
- Pod n
- Raw predicted metrics
Node 1
- Node 2
- Node n
- Raw predicted metrics


Alameda workflow
kubectl
Kubernetes/etcd
operator
alameda-ai
recommender
evictioner
admission controller
executor
datahub
Prometheus
Influxdb
User selects pods to be managed
create alamedascaler CR
an alamedascaler CR is created
register Pod for prediction and recommendation
write Pod identification to Influxdb
when alameda-ai is triggered
Alameda generates prediction
get lists for prediction
retrieve lists
get historical metrics
retrieve historical metrics
write prediction metrics
write prediction metrics
when recommender is triggered
Alameda generates recommendations
get lists for recommendations
retrieve lists
get predicted metrics
retrieve predicted metrics
write recommendations
write recommendations
write alamedarecommendation CR
Alameda polls for new recommendations to execute container resizing
get latest recommendations
get latest recommendations
delete pods which recommendations suggest to change resource limit/request
call mutatingwebhook for a newly create pod
get recommendation for the pod to be created
get recommendation for the pod to be created
return mutated resource limit/request recommended by Alameda
Alameda polls for new recommendations to execute pod rescaling
get latest recommendations
get latest recommendations
change pod replica


Recommendation Execution Workflow
datahub
evictioner
admission controller
kubernetes
Evictioner chooses pods for resource request/limit updation
get new recommendations
loop
[for each pod in recommendations]
pod need to update its resource? (*1)
delete pod if it needs updation
after some time
admission controller updates resource request/limit
create a new pod' for the deleted pod
send pod' spec to admission controller
get recommendations for pod' (*2)
mutate resource request and limit in pod' spec
create pod' instance


Control Flow to Choose Pods for Eviction
more pods with new recommendations?
false
true
current request of pod is available?
true
false
recommended request * (1 - factor) < current request < recommended request * (1 + factor)
true
false
evict pod
evict pod


Control Flow to Assign Recommendation for a Pod Spec
fetch recommendations that are for this pod Spec
filter out recommendations which are for running pods
filter out recommendations which are already assigned to pod creation request
mutate Spec with one of the remaining recommendations

参考

- https://pkg.go.dev/github.com/containers-ai/alameda

参考

- https://github.com/umbertogriffo/Predictive-Maintenance-using-LSTM
- https://github.com/AICoE/prometheus-anomaly-detector
- https://github.com/nfrumkin/forecast-prometheus
- https://www.slideshare.net/GeorgOettl/analyze-prometheus-metrics-like-a-data-scientist
- https://www.ericsson.com/en/blog/2019/6/automated-fault-management-machine-learning
- https://github.com/VictoriaMetrics/VictoriaMetrics
- https://github.com/nfrumkin/forecast-prometheus
- 《*Distributed Systems Observability*》
- https://dbaplus.cn/news-134-3008-1.html
- https://mc.ai/review-of-fault-management-with-machine-learning-techniques/
- https://medium.com/@prasad.kumkar/review-of-fault-management-with-machine-learning-techniques-5d603fbbfef5
- https://next.redhat.com/2019/11/18/prometheus-anomaly-detection/