

基于演化博弈的弹性调控平台任务调度研究

熊 文¹, 于全喜², 吴任博¹, 伦惠勤¹, 孔海斌², 谭军光²

(1. 广州供电局有限公司 电力调度控制中心, 广州 510620; 2. 东方电子股份有限公司 技术中心, 山东 烟台 264000)

摘 要: 针对电网生产控制云 PaaS 类弹性调控平台上任务调度性能波动大的问题, 构建包含节点感知器、资源状态服务器和任务调度器等核心构件的任务调度框架。在模型选择阶段, 采用混合博弈法, 根据任务对不同资源的偏好编排执行节点, 完成节点负载预估计算。在模型突变阶段, 分析任务执行效果调整其资源分配, 获得具有较高节点评分的任务调度策略, 指导后续任务的博弈节点选择。在分布式监视控制与数据采集系统上进行任务调度框架的测试验证, 实现了 7 个 ~ 25 个分片、500 万量测点级的任务负载均衡和容灾处理, 结果表明基于演化博弈的任务调度策略相比开源任务调度工具性能更加稳定。

关键词: 演化博弈模型; 平台即服务; 分布式任务调度; 目录服务; Docker 容器引擎

中文引用格式: 熊文, 于全喜, 吴任博, 等. 基于演化博弈的弹性调控平台任务调度研究[J]. 计算机工程, 2019, 45(7): 86-94.

英文引用格式: XIONG Wen, YU Quanxi, WU Renbo, et al. Research on task scheduling of flexible regulation and control platform based on evolutionary game[J]. Computer Engineering, 2019, 45(7): 86-94.

Research on Task Scheduling of Flexible Regulation and Control Platform Based on Evolutionary Game

XIONG Wen¹, YU Quanxi², WU Renbo¹, LUN Huiqin¹, KONG Haibin², TAN Junguang²

(1. Power Dispatching and Control Center, Guangzhou Power Supply Bureau Co., Ltd., Guangzhou 510620, China;

2. Technology Center, Dongfang Electronics Co., Ltd., Yantai, Shandong 264000, China)

[Abstract] Aiming at the problem of fluctuation of task scheduling performance on the Platform as a Service (PaaS) flexible regulate and control platform of the power grid production control cloud, a task scheduling framework including core components such as Node Perceptron (NP), Plat Resource Status Server (PRSS) and Task Scheduler (TS) is constructed. In the model selection stage, the hybrid game method is adopted, and the execution nodes are arranged according to the preference of the tasks for different resources, to complete the node load estimation calculation. In the abrupt change stage of the model, the execution effects of the tasks are analyzed to adjust its resource allocation, obtain a task scheduling strategy with higher node scores, and guide the selection of game nodes for subsequent tasks. The task scheduling framework is tested and verified on the distributed Supervisory Control and Data Acquisition (SCADA) system, and the task load balancing and disaster recovery processing of 7 ~ 25 fragments and 5 million measurement points are realized. Results show that the task scheduling strategy based on evolutionary game has more stable scheduling performance than the open source task scheduling tool.

[Key words] evolutionary game model; Platform as a Service (PaaS); distributed task scheduling; directory service; Docker container engine

DOI: 10.19678/j.issn.1000-3428.0051836

0 概述

依托云计算、大数据等先进 IT 技术, 改进现有智能电网调控基础平台, 研发面向 202x 年的弹性调

控平台, 已成为业界的发展趋势。文献[1]给出分布式监视控制与数据采集 (Supervisory Control and Data Acquisition, SCADA) 系统架构, 描述了分布式资源和任务管理、实时数据管理和计算处理方法。

基金项目: 南方电网公司广州供电局科技项目“基于云技术的调度控制系统支撑框架和处理模式研究”(GZHKJXM20160006)。

作者简介: 熊 文 (1973—), 男, 高级工程师、硕士, 主研方向为电力系统调度; 于全喜 (通信作者), 工程师、硕士; 吴任博、伦惠勤, 高级工程师; 孔海斌、谭军光, 硕士。

收稿日期: 2018-06-15

修回日期: 2018-07-30

E-mail: ximing224@163.com

文献[2]论证了调度自动化系统应用云技术的可行性,提出一种基于云计算平台的调度自动化系统架构。弹性调控平台是支撑电网调度自动化系统的PaaS实现,需要设计良好的任务调度策略,提供稳定的云计算服务。

文献[3]综合考虑云计算环境下资源节点与通信链路的可靠性问题,设计基于遗传算法的可信云资源调度算法。文献[4]提出一种网格联合调度策略,减少了虚拟机抢占的数量。文献[5]提出一种基于OpenStack的快速选择调度器。文献[6]结合能量和性能感知的调度策略,开发一套性能和能量感知的资源分配、任务调度工具。然而,以上研究均未充分考虑实时数据处理类系统的运行特点,且任务调度需综合考虑节点的计算、存储、网络等资源供给情况。文献[7]利用博弈理论将调度节点比作博弈的参与者,综合考虑节点资源和网络性能,采用混合博弈策略实现Nash均衡。文献[8]中的演化博弈适合解决网络中的动态博弈问题,描述了网络行为和群体行为的评价指标、分析模型及评价框架。

考虑到弹性调控平台的资源异构性、伸缩性及性能波动性等问题,任务调度时可在不完全信息条件下演化博弈竞争资源。本文基于演化博弈模型构建包含节点感知器(Node Perceptron, NP)、资源状态服务器(Plat Resource Status Server, PRSS)和任务调度器(Task Scheduler, TS)3个核心构件的任务调度框架。

1 演化博弈模型

本文演化博弈研究的对象是随时间变化的虚拟节点集群,分析节点行为评分计算、微服务选择运行节点的任务调度技术。文献[8]随机演化博弈的建模过程具体如下:

1)基本参数设定:决策者是TS,决策集包含节点感知和任务指令模型,收益矩阵是节点的行为评分。当决策策略有限时,所有可能的策略组合可能只收益一种可用矩阵表示的结果,可用简单的函数表示。

2)变更决策策略:突变阶段TS尝试新的任务编排策略(执行效果可能较差),通过不断排除错误,获得的较高评分策略将被其他节点采用并计算其行为评分。

3)参数求解:求解演化博弈过程中设置的一些参数,如服务处理的测量点数量、权重调节因子等。

4)均衡策略求解:结合平台任务调度的特点,采用资源性能排序和累计计算方法,不断修正节点评分策略。

模型选择阶段采用混合博弈法,根据任务对不同资源的偏好编排执行节点,完成节点负载预估及行为评分计算。模型突变阶段分析任务的执行效果,调整节点上任务的资源分配,获得较高的节点评分策略,逐步进化构成演化博弈。

1.1 节点感知模型

文献[9]提出资源感知调度算法,将资源分为硬约束(内存)与软约束(CPU和网络),设计一种Storm环境下基于权重的任务调度算法。文献[10]研究异构Storm环境下的任务迁移策略,根据节点CPU、内存和网络带宽的负载情况以及各类资源的优先级顺序,开展节点选择和任务迁移。本文节点感知模型选择平台关注软硬约束的关键资源,键值模式量化设计如下:

1) domain

业务域D: <"D-" + name, {name, desc}>;键为"D-"字符串接名称,值的属性为名称和描述。

2) class

资源类C: <"C-" + name, {name, desc, method}>;method为类中对象object的原型方法。典型类如下:(1) <"C-Node", {"Node" "节点", [host, Label, store, cpu, mem, disk]}>;store = {calc, storage, net, all},依次为计算、存储、网络及综合评分。(2) <"C-Base", {"Base" "基础服务", [weight, status, cpu, mem]}>。(3) <"C-App", {"App" "应用服务", [weight, status, cpu, mem, type, flow]}>。(4) <"C-Net", {"Net" "网络", [TX, RX, peak, rates]}>。

3) object

资源对象O: <"O-" + D.name + ":" + C.name + ":" + name, {oid, name, desc, register, uptime, mask}>;键为"O-域名:类名:对象名";值中oid为UUID,register = {md5, partition, passwd}为对象注册信息(属性依次为MD5码、参数分片号、私有密码),uptime为最后更新的标准秒值(到1970年1月1日的秒数),mask为状态掩码和操作类别。

4) attributes

资源对象属性A: <"A-" + O.oid + ":" + C.method_n, {[info, uptime, value]}>, C.method_n对应资源类中的method(n是位次编号),info存储键值信息(空用"-"),uptime是标准秒值,value是浮点型数据(空用"0")。

1.2 任务指令模型

文献[11]的ACOHs算法定义了六元组任务,使用队列优化任务的选择策略。本文设计将任务指令化处理,任务TASK = {task, target, status}:属性

task 为任务指令;任务标签 $\text{target} = \{\text{calc}, \text{storage}, \text{net}\}$, 依次为计算量、存储数据量、网络流量;任务状态 status : -1 为异常, 0 为待校核, 1 为常规, 2 为执行校核。task 设计如下:

```
task = {# 定义 n 为资源对象的名称
  "Domain": {"create n" "remove n" "show n"}
  "Node": {"create nLabel" "remove n" "update
n Label_old Label_new" "show n"}, #节点
  "Base": {"create N n" "remove N n" "update
N n_old n_new" "show N n"}, #基础服务, 定义 N
为节点名
  "App": {"create app N n" "remove app N n"
"update app N n_old n_new" "show app N n"}, #业
务应用服务
  ...}
```

执行节点 create 指令时自动下载业务标签 Label (scada|fert|alarm|app|model|pmi|webd|admin, 用 | 分隔标示业务) 指定的 Docker images (存在则不下载), 启动 Base 类服务。基础服务指令包含 "knoded" "kprssd" "ktskd" 等, 分别为 NP、PRSS 和 TS 等服务。业务应用基命令 app 包含 "kscadad" "kcontrold" "kalmapid" "kmodeld" "krtud" 等, 对象名称格式为 k_app_分片编号_主备编号, 如 k_kscadad_001_0 是处理模型 001 号分片 SCADA 主服务 (主备编号: 0 为主服务, 备服务按正整数编号)。

1.3 作业博弈

作业 (job) 是为完成某项功能而提交到 TS 的任务队列 (FIFS), $\text{job} = \{\text{name}, \text{taskQueue}\}$ 。作业博弈时计算每个节点的等待执行任务数和负载值, 重新编排超配置负载节点上的新增任务。

1) 作业平均偏好和任务计算偏差

(1) job.taskQueue 中有 n 个任务, 取 RSDB 库中同域内的 App 任务集 (总数 h), 计算作业平均偏好:

$$\text{job.ave}(\text{calc}) = \frac{\sum_{i=1}^n \text{taskQueue}(i). \text{target. calc} + \sum_{i=1}^h \text{RSDB}(i). \text{target. calc}}{n + h} \quad (1)$$

$\text{job.ave}(\text{storage})$ 和 $\text{job.ave}(\text{net})$ 的计算公式类似于式 (1)。

(2) 取队列中的任一任务 p , 计算偏差指数:

$$p. \text{bias}(\text{calc}) = \frac{p. \text{target. calc} - \text{job.ave}(\text{calc})}{\text{job.ave}(\text{calc})} \quad (2)$$

$p. \text{bias}(\text{storage})$ 和 $p. \text{bias}(\text{net})$ 的计算公式类似于式 (2)。任务 p 的 calc 、 storage 或 net 偏差指数高

于其对应的平均偏好值越多, 其偏离程度正向越高, 说明任务 p 对该项偏好越明显, 要求部署节点的对应指标评分越高。

2) 负载预估及校核

节点上已经运行 i 个业务服务时的平均偏好为 $\text{job.ave}(\text{calc})$, 执行计划新增 j 个任务, 负载预估计算:

$$\text{load}(\text{calc}) = \frac{\text{job.ave}(\text{calc}) \times (i + j)}{i} \quad (3)$$

$\text{load}(\text{storage})$ 和 $\text{load}(\text{net})$ 的计算公式类似于式 (3)。当各行为均不超过配置范围时, 可发布编排任务。如果某项负载超过范围, 则需要把负载高的节点上该项目偏差指数最大任务重新编排给其他节点。

1.4 资源对象博弈

资源对象博弈的目的在于使每一个任务竞争到符合其偏好的资源, 通过计算每个节点的评分, 不断演化修正评分策略, 提高任务编排质量。设域中有 m 个节点。

1) 节点综合评分

为简化计算, 分析节点感知模型中 cpu 、 mem 、 disk 、 TX 及 RX 等关键资源的排序位置, 计算节点的行为评分:

(1) 服务启动时 Lookup 检索 Node 类节点, 再按节点检索属性值; 启动后调用 Listen 获取属性变化信息。

(2) 按属性值从大到小排序, 记录节点排序位置 Sort 。

(3) 计算节点 n 的评分值:

$$n. \text{calcScore} = \frac{100 \times \sqrt{\text{Sort}(n. \text{cpu})^2 + \text{Sort}(n. \text{mem})^2}}{2 \times m} \quad (4)$$

$n. \text{storageScore} =$

$$\frac{100 \times \sqrt{\text{Sort}(n. \text{mem})^2 + \text{Sort}(n. "/>$$

$$n. \text{netScore} = \frac{100 \times \sqrt{\text{Sort}(n. \text{Net. TX})^2 + \text{Sort}(n. \text{Net. RX})^2}}{2 \times m} \quad (6)$$

$$n. \text{allScore} = \sqrt{n. \text{calcScore}^2 + n. \text{storageScore}^2 + n. \text{netScore}^2} \quad (7)$$

任务可根据偏好首先编排给节点标签组内对应行为打分最高的节点, 如果审核异常, 则编排给打分次高的节点。任务无特殊要求, 按节点综合评分值编排。

2) 节点性能指数

第 k 个节点 n_k 性能指数公式为:

$$n_k.perf(calc) = \frac{n_k.store.calc - \frac{\sum_{i=1}^m node(i).store.calc}{m}}{\frac{\sum_{i=1}^m node(i).store.calc}{m}} \quad (8)$$

$storage$ 和 net 的性能指数计算公式类似于式(8)。

$$n_k.perf(all) = \sqrt{n_k.perf(calc)^2 + n_k.perf(storage)^2 + n_k.perf(net)^2} \quad (9)$$

当某项性能指数正值越大时,说明该节点资源性能高于平均水平越显著。

3) 任务满意度

当队列中有 n 个任务时,取一任务 p 的执行节点评分、任务偏差指数从大到小排序所处位置 $sort_bias$ 、节点性能指数从大到小排序所处位置 $sort_perf$ 计算 p 满意度:

$$p.satisfy(calc) = a \times (pnodecalcStore - \frac{\sum_{i=1}^m node(i)storecalc}{m})^2 + b \times (\frac{sort_bias(calc)}{n} - \frac{sort_perf(calc)}{m})^2 \quad (10)$$

其中, a 、 b 为权重调节因子(当调整满意度到合理值后,固定 a 和 b 的值), $storage$ 和 net 的任务满意度计算公式类似。分析式(10),将某项性能排名靠前的资源分配给所有任务中对这项性能有较高要求的任务,若符合任务调度的需求,则满意度较高。

1.5 博弈反馈及均衡

节点资源对微服务的计算、存储或网络行为的偏好,比其他运行微服务更具适应性,并且种群会更趋向于由其单独构成,这样的演化稳定结果即单态型,称为演化稳定策略(Evolutionary Stable Strategy, ESS)。本文中的计算、存储、网络等资源采用单态性演化,所占用的资源状态相互作用构成矩阵,可逐步进化构成一个演化博弈。

设节点 n 上累计执行的任务数为 k ,前期 $calc$ 评分为 $n.calcScore1$ (原始评分可初始化为0),本次任务调度后的评分为 $n.calcScore2$,则本次调度获得的 $calc$ 评分对其评分的演化修正公式为:

$$n.calcScore = \frac{k \times n.calcScore1 + n.calcScore2}{k + 1} \quad (11)$$

$storage$ 和 net 的评分演化修正公式类似于式(11)。

节点可部署的并行执行同一业务服务的主备分片应用个数的限制公式为:

$$main_{num} = \left\lceil \frac{\text{有效服务数}}{\text{节点个数}} \right\rceil$$

$$backup_{num} = main_{num} + 1 \quad (12)$$

节点会依据自身执行任务后的评分改进任务竞争能力。每个节点都会根据其自身资源的特点,影响评分走势,节点资源对象的种群稳定结构即达到博弈的均衡。

1.6 任务调度算法

弹性调控平台上任务调度的服务组件架构设计具体如下:

```

Begin 选取业务域;
节点感知、采集信息存储到键值数据库中;
按计算、存储、网络等资源训练排序,初始化节点综合评分;
资源对象博弈组件 {
    监视域范围内节点资源信息,按感知模型更新键值数据库;
    按域中资源对象分类排序,记录排序位置;
    计算节点分类评分及综合评分,分值更新键值数据库;
}
作业博弈组件 {
    接收新作业 job;
    while( 取待调度的任务 p, p ∈ job.taskQueue ) {
        计算 p 的偏差指数,确定 p 偏好的资源;
        综合评分最高的节点竞争到 p,预估 p 执行后节点上计算、存储、网络 3 类负载是否超过阈值;
        未超过阈值时用评分最高的节点名组装任务指令 p.task;
        超过阈值时依次取评分次高的节点竞争 p,直到 p 被分配;
        发布 job 的执行计划,目标节点获取后执行并反馈结果;
    }
}
博弈反馈组件 {
    收集执行计划的反馈信息,执行异常的任务需再重发 2 次;
    取键值数据库的评分值,估算任务满意度的权重调节因子 a 值和 b 值,计算满意度;权重因子核校稳定后,固定不变;
    后续任务计算的满意度高,任务的节点调度分配合理;否则,修正对应的评分式(4)~式(6),获取高满意度;
}
博弈均衡组件 {
    设置突变条件:参考式(12)限制主备分片应用的个数;
    取键值数据库 APP 类的 object,统计超编任务集;
    while( 取任务 p, p ∈ 超编任务集 ) {

```

根据 p 识别业务的服务组,确定可调度任务的节点组;

检索出节点组内运行主服务最少的节点 a 和运行备服务最少的节点 b;p 的操作对象是主服务,p 分配给 a 节点;p 的操作对象是备服务,p 分配给 b 节点;}

发布突变执行计划;

取节点组中节点的评分值,评分表现均趋于平均,则得到博弈的均衡;否则说明本次尝试存在缺陷,可修正突变条件重新尝试;

}

End

2 任务调度框架

本文通过资源对象及属性变化队列 OAQueue 和任务调度等待队列 SWQueue 2 个弹性队列,完成集群节点间的消息通信,任务调度框架设计如图 1 所示。

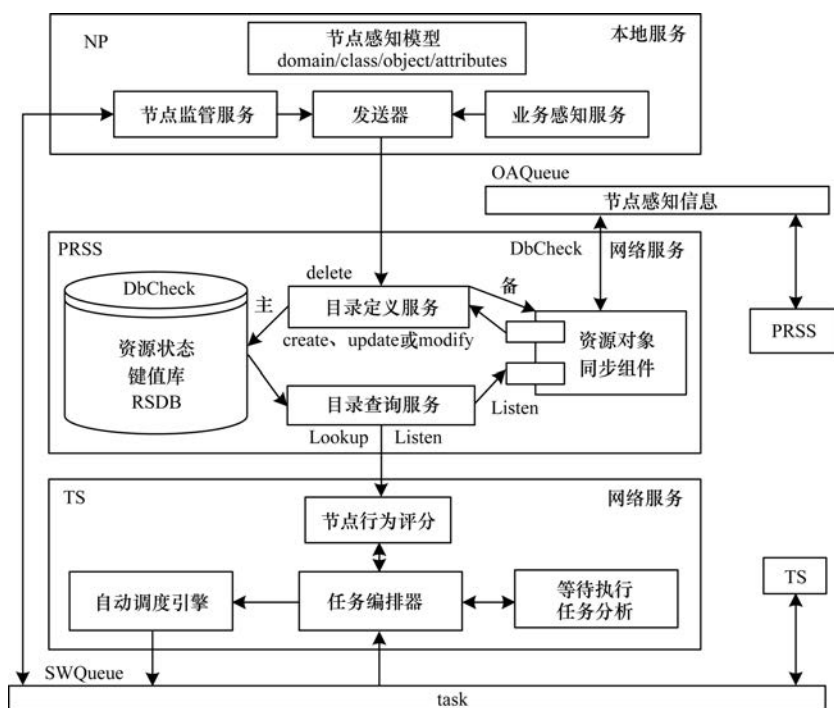


图 1 弹性调控平台任务调度框架

2.1 节点感知器

节点感知器内部组件包含节点监管服务、业务感知服务和发送器。节点监管服务用于采集节点的 CPU、内存、磁盘、网络流量、关键进程等资源信息,并订阅、执行 SWQueue 的任务指令。业务感知前置通道状态及流量、事件告警及弹性消息总线流量等网络信息。发送器收集感知结果,发送给节点的 PRSS。

2.2 资源状态服务器

PRSS 采用一主多备模式运行,内置 RSDB 键值数据库存储感知信息,提供目录定义和查询服务,通过资源对象同步组件与 OAQueue 交互消息。目录查询 Listen 服务识别主服务器的定义成功信号,将事务通知节点的资源对象同步组件和 TS。目录查询 Lookup 服务提供调用节点 RSDB 的实时检索功能,处理流程如下:

1) 主 PRSS 接收本地定义事务(create、delete、

update 或 modify)或订阅 OAQueue 上标志为 0 的事务,直接维护本地 RSDB;Listen 获取资源对象的变化事务(标志为 1)转发 OAQueue;主 PRSS 每执行“配置指定条数”的定义事务,执行一次 DbCheck 操作。

2) 备 PRSS 收到本地定义事务(标志为 0)转发给 OAQueue;订阅 OAQueue 上标志为 1 的事务更新本地 RSDB;转发的事务被订阅执行,反馈正确通知。

3) 事务的发起节点若是主服务器,则发起者得到 RSDB 处理通知;发起节点若是备服务器,则发起者得到反馈通知。

2.3 任务调度器

TS 处理任务调度算法组装的任务调度指令,采用一主多备的网络部署模式运行,包含 4 个核心组件:节点行为评分,等待执行任务分析,任务编排器和自动调度引擎,工作流程如图 2 所示。

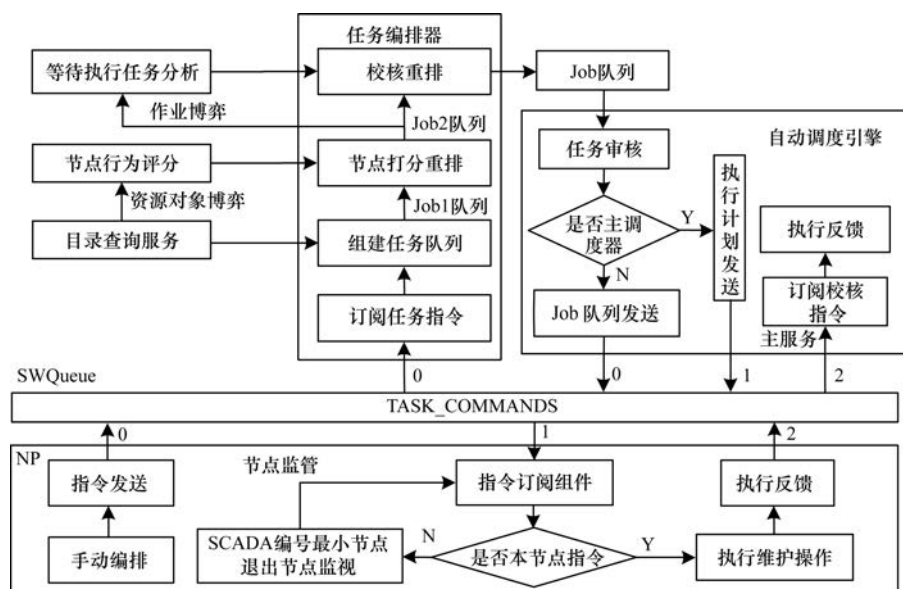


图2 任务调度器工作流程

任务调度器的具体步骤如下:

1) 调度器启动后,目录查询服务定时 Lookup 查询节点的资源信息按演化博弈模型的计算节点评分值,提交节点行为评分组件;通过 Listen 监听资源的变化信息,按博弈反馈及均衡算法,不断修正节点评分。

2) 订阅任务指令组件消费待校核的任务指令,提交给组建任务队列组件校核,构建 FIFO 的 Job1 队列。

3) 节点打分重排,Job1 校核转存 Job2 队列:

(1) 只编排 App 类 create、remove 或 update 指令。

(2) 取 Job1 中一 App 类任务,编排给业务组中评分最高的节点;如果是备服务,则其对应的主服务在编排节点上,需编排给评分次高的节点;结果输入 Job2。

(3) 返回步骤(2),编排后续任务,直到 Job1 处理完毕。

4) 等待执行任务分析和校核重排:

(1) Lookup 查询域中每个节点上运行的微服务个数。

(2) 遍历 Job2,统计每个节点上待执行的任务个数。

(3) 负载预估,将负载越限的任务重新编排给评分次高的节点;无负载过高的节点时将 Job2 赋给 Job 队列。

5) 自动调度引擎接收 Job 队列,审核指令,判断 TS 是否为主服务,若是则将 Job 转换为执行计划(状态变为 1)发送出去,否则直接把 Job 发送给 SWQueue。

6) 节点监管服务的指令订阅组件,消费标志为

1 的指令;判别是否为本节点投运的指令,若是,则执行指令,并维护 RSDB;执行后将指令状态改为 2 后提交 SWQueue。如果是节点退出类任务,则由 SCADA 编号最小的节点完成目录定义操作。

7) 主调度器自动调度引擎的订阅校核指令组件,执行 SWQueue 处理标志为 2 的指令,完成闭环验证。

8) 如果常规指令发送 20 s 后,主调度器未收到反馈,则本次任务调度失败,生成告警信息。当告警确认时,重新编排失败的指令,提交“手动编排”组件处理。

2.4 DbCheck

DbCheck 依据对象的 update 时间实现校核:

1) 主 PRSS 启动 Lookup 查询出"O-" + D. name 开头的指定 domain 的 object;资源对象同步服务封装 object 成待校核类事务(标志为 2)提交给 OAQueue。

2) 备 PRSS 订阅待校核事务,得到集合 CheckObj,校核步骤具体如下:

(1) 取 CheckObj 中的一个待校核对象 obj。

(2) 按 obj. key 查询本地 RSDB,若不存在 obj,则执行 create,并记录日志;若 key 存在,则读取对应的对象 obj_local。

(3) 更新时间比较:若 obj_local. uptime == obj. uptime,则返回步骤(1);如果 obj_local. uptime < obj. uptime,则表示本地资源对象存在延迟,通过远程调阅接口从主 PRSS 的 RSDB 下载 obj 对应的属性信息集,并在本地执行 modify 操作;如果 obj_local. uptime > obj. uptime,则表示 obj 所执行的节点之前为主节点,且变化数据未发布,将 obj_local 打上 modify 标志(标志为 1)后提交给 OAQueue。

3) 备 PRSS 订阅 OAQueue 上标志为 1 的事务, 本地执行定义操作。

3 框架应用及仿真

在广州供电局基于 OpenStack 与 Docker 技术^[12]搭建的 EMS 原型系统上开展图 1 所示框架的测试验证, 设原型中每个分片关联一组 SCADA 应用服务、加载 20 万个量测点。原型 IaaS 平台提供固定的虚拟节点资源 (10 核 CPU、64 GB 内存), 其资源调度问题可参考文献[13]。通过测试 7 个 ~ 25 个并行计算 SCADA 分片的任务调度性能, 验证了本文调度框架对 500 万个量测点的实时数据处理的可行性。

分析图 2 可知, 任务调度器采用 FIFO 队列调度模式, 任务指令的优先级: Node > Base > App, 高等级 task 指令执行完毕后才能执行低等级的指令, 同等级的指令可以并行调度。本文任务指令模型封装了单机版 Docker 运维操作命令, 依赖任务调度框架实现容器调度。仿真实验对比了测试环境与 E8000 平台网络对象和 Docker Swarm 任务调度的优缺点, 实现负载标准差和时间跨度值的可比性验证。

3.1 框架应用

图 1 的任务调度框架采用演化博弈模型及云技术改进了 E8000 平台的网络对象感知调度技术, 通过分析 SCADA 任务编排、主备服务异常、节点退出等核心业务应用, 描述弹性调控平台业务应用的通用调度规则。

1) SCADA 任务编排

设 scada 标签配置总服务数为 25、有效数为 7、一主一备模式部署, 测试 4 个节点 (传统物理机一般部署 4 个节点) 打分排序为 scada04 > scada03 > scada02 > scada01。参考式 (12), 每个节点至多运行 2 个主服务或 3 个备服务。主服务按节点打分值从大到小的顺序部署, 打分最高的节点编排上主服务后变为打分最低的节点; 备服务按节点打分值从大到小的逆序部署。SCADA 初始编排 (服务前缀为 k_kscadad_) 如表 1 所示。

表 1 SCADA 初始编排

节点	主服务	1 号备服务
scada01	004_0	001_1, 005_1
scada02	003_0, 007_0	004_1
scada03	002_0, 006_0	003_1, 007_1
scada04	001_0, 005_0	002_1, 006_1

作业创建及执行过程:

(1) ktaskd Domain create EMSI; #创建域

(2) 按 SCADA 组打分排序的顺序, 创建虚拟机节点:

ktaskd Node create scada04 SCADA; ...

(3) 按表 1 先主后备服务的顺序生成启动指令集:

ktaskd App create kscadad scada04 k_kscadad_001_0; ...

ktaskd App create kscadad scada01 k_kscadad_005_1;

(4) 将以上任务指令 (状态为 0) 写入队列。参考图 2, 完成任务调度。

scada01 的 Node 类感知资源如图 3 所示。

```
object:
  "O-EMSI:Node:scada01", {"oid":"113152940654461522", "name":"scada01", "desc":"scada01", "register":{"md5":"6d22e86f392d2e1258a1e88df8ed0444", "partition":"-", "password":"ems123"}, "uptime":1526572772, "mask":"0x1112"}
attributes:
  <"A-113152940654461522:host", {"info":"scada01", "uptime":"1526572772", "value":0}>;
  <"A-113152940654461522:label", {"info":"scada01", "uptime":"1526572772", "value":0}>;
  <"A-113152940654461522:store", {"info":{"key":"calc", "value":78}, {"key":"storage", "value":56}, {"key":"net", "value":84}, {"key":"other", "value":75}}, "uptime":"1526572772", "value":0}>;
  <"A-113152940654461522:cpu", {"info":"-", "uptime":"1526572772", "value":0.32}>;
  <"A-113152940654461522:mem", {"info":"-", "uptime":"1526572772", "value":0.82}>;
  <"A-113152940654461522:disk", {"info":{"key":"/", "value":0.18}, {"key":"/data", "value":0.26}, {"key":"/usr", "value":0.6811, "uptime":"1526572772", "value":0}>;
```

图 3 scada01 的 Node 类感知资源

在节点新增 scada 标签后, 计算每个节点运行的最大主服务和备服务数。超过服务限制个数的节点将负载最小的主服务或负载最大的备服务, 迁移到新节点上。

2) 主服务异常

Listen 检测到 Base 类服务异常, 在本节点重启 3 次, 启动异常发出告警; 检测到 App 类主服务异常, 将备服务变更为主服务, 并在组内其他节点上启动变更成功的备服务。若 k_kscadad_001_0 异常, 则调度过程如下:

(1) 检索 k_kscadad_001_0 服务是否存在, 若不存在, 则此服务正常关闭; 若 "O-EMSI: App: scada04:k_kscadad_001_0" 存在, 则 scada04 上服务异常, 需移除:

ktaskd App remove kscadad scada04 k_kscadad_001_0;

(2) 检索出其备服务集, 若 k_kscadad_001_1 存在, 则:

ktaskd App update kscadad scada01 k_kscadad_001_1 k_kscadad_001_0;

若 1 号备服务变主服务失败, 则向后变更 2 号; 以此类推, 直到主服务启动成功。

(3) 001_1 备服务编排给 scada 组内剩余节点评分最高的节点 (scada04), 若编排满足要求, 则执行:

ktaskd App create kscadad scada04 k_kscadad_001_1;

(4) 任务指令队列发送给 SWQueue。

3) 备服务异常

Listen 检测到 App 类备服务异常, 重启 1 次仍失败则需编排任务, 如 scada01 上 k_kscadad_001_1 异常:

(1) 检索 "O-EMSI: App: scada01:k_kscadad_001_1", 若不存在, 则服务正常关闭; 否则服务异常, 先移除服务:

ktaskd App remove kscadad scada01 k_kscadad_001_1;

(2) 生成启动指令:

ktaskd App create kscadad scada01 k_kscadad_001_1;

(3) 任务指令队列发送给 SWQueue。

(4) 如果 20 s 后,服务还未启动,选择新节点 (scada03 无 k_kscadad_001_0,且评分最高) 执行启动指令:

```
ktaskd App create kscadad scada03 k_kscadad_001_1;
```

(5) 任务指令队列发送给 SWQueue。

4) 节点退出

SCADA 编号最小的运行节点检测到节点 n 异常或退出,需把 n 上的 App 类服务迁移到其他节点上,具体步骤如下:

(1) Lookup 检索 "O-EMSI: App: n. name:", 生成应用服务对象集合 Obj。

(2) Obj 中取一个对象 obj,找到 obj 的基命令。

(3) obj 是主服务,按"主服务异常"处理。

(4) obj 是备服务,按"备服务异常"处理。

(5) 收集执行失败的指令,返回步骤(3),直到无失败反馈。

3.2 仿真及对比测试

E8000 网络对象技术是一种传统的分布式管理技术,将分布式系统中的节点、平台或应用服务、重点业务应用等抽象成基于目录服务统一管理的对象,感知对象的有或无,采用 Leader 选举算法实现主备服务的管理。Docker Engine 中 Swarm 提供服务发现、负载均衡、扩展和安全性,文献[14]对 Docker-Swarm 进行改进,提高资源利用率、集群负载均衡及稳定性。容器调度需考虑部署和迁移过程中服务的参数加载问题。文献[15]提出的云平台资源弹性调度策略,综合考虑数据存储、资源负载和应用服务性能,比较了 Swarm_opt 和 Swarm 的编排性能。

本文在相同的业务应用场景下,首先开展单个分片的主备服务任务调度实验,完成容灾耗时对比,然后增加负载及应用服务数进行负载波动标准差对比,最后扩展业务分片、增加计算服务,完成演化调度与 Swarm 的时间跨度对比。加载一个 20 万量测点的 SCADA 分片,在 4 个 SCADA 节点轻负载下逐个停止 "kscadad" "kcontrolld" "kalmapid" "kmodeld" "krtud" 5 个主服务(一主三备部署模式),3 种调度策略的主备迁移容灾耗时对比结果如图 4 所示。

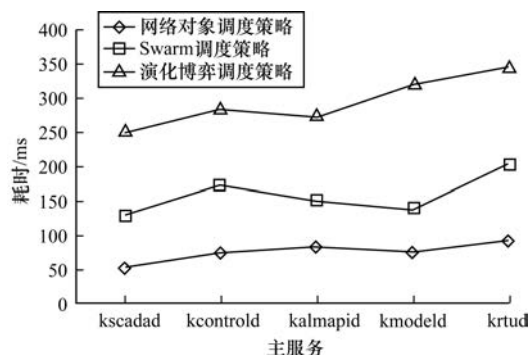


图 4 3 种调度策略的主备迁移容灾耗时对比结果

网络对象调度只需判断对象的有无,开展分布式 Leader 选举,效率最高,Swarm 调度嵌入比较大的调度框架,而演化博弈调度则经历了更复杂的排序评分算法,容灾效率稍差。网络对象调度只支持单分片模式,增加 SCADA 分片容量到 120 万量测点,4 个 SCADA 节点重负载下运行 20 个业务应用服务(主备模式:SCADA 服务一主三备,前置 4 个服务并行采集,其他服务一主一备)。依次采用 3 种调度策略,完成容灾比对 5 个服务的主、备异常的任务调度,计算每轮调度后节点 CPU 和内存负载的标准差为其负载波动值。在 10 次任务调度后,节点上 CPU 标准差计算方法具体如下:

$$\bar{x} = \frac{1}{n} \sum_{k=1}^n cpu(k)$$

$$\sigma(CPU) = \sqrt{\frac{1}{n} \sum_{k=1}^n (cpu(k) - \bar{x})^2}$$

其中, $n=10$, $cpu(k)$ 为第 k 个任务调度后的 CPU 使用率乘以 100;内存标准差按使用率求解。单分片模式各节点上运行的业务应用服务相同,负载基本均衡。

在图 5 中,3 种调度策略的负载波动基本持平。主服务需校核模型,完成业务计算、组播计算,资源消耗多,切换执行节点会导致其负载波动明显。当调度多分片的业务服务时,文献[13-14]采用 Swarm 固有的任务编排策略与演化博弈调度策略进行对比。按表 1 初始化编排,选用 Swarm-spread 策略与图 2 的调度器进行比较完成仿真实验。初始 SCADA 节点排序及评分如表 2、表 3 所示。

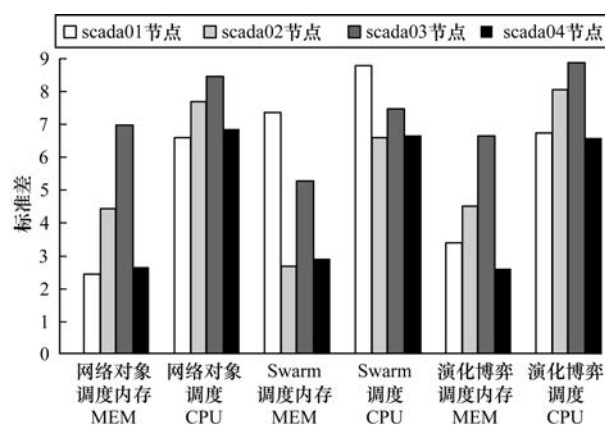


图 5 3 种调度策略的负载波动标准差对比结果

表 2 SCADA 节点感知模型中关键资源排序结果

节点	cpu	mem	disk1	disk2	TX	RX
scada01	4	2	3	4	3	4
scada02	3	1	4	3	1	2
scada03	2	3	1	2	2	3
scada04	1	4	2	1	4	1

表 3 SCADA 节点行为评分结果

节点	calcScore	storageScore	netScore	allScore
scada01	55.9	44.9	62.5	95.1
scada02	39.5	42.5	28.0	64.4
scada03	45.1	31.2	45.1	71.0
scada04	37.5	38.2	51.5	74.3

此时,将节点的有效分片数从 7 个逐步扩展到 25 个,平台量测点从 140 万逐步扩展到 500 万。顺序完成 5 次不同分片的主、备 SCADA 服务调度,时间跨度对比结果如图 6 所示。

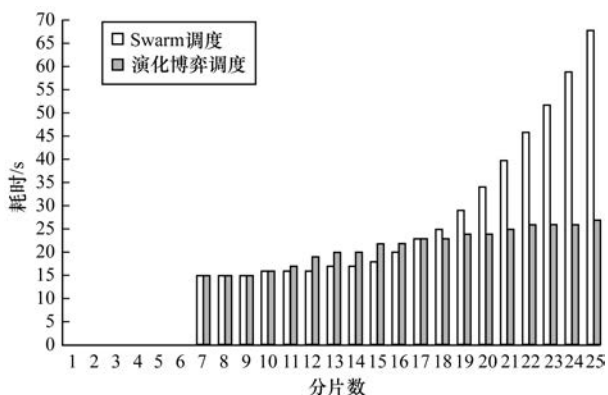


图 6 分片扩展的时间跨度对比结果

多分片服务的关联服务及模型加载校核耗时较多。Swarm 调度时关注 CPU 和内存空闲度,经历 10 次调度后,在 16 个分片时主备服务分布不均衡(存在节点运行 2 个主服务、11 个备服务的情况),主备负载均衡效果差、忽视了网络因素。随着实验分片的增加,节点资源负载逐步接近告警值,Swarm 片面均衡调度因素的影响越来越明显,总执行时间呈指数增长趋势。

演化博弈调度综合计算节点的计算、存储、网络资源信息,给出节点合理的评分后按业务组开展任务调度,通过任务选择执行节点、节点负载预估、任务执行博弈反馈等手段,不断进行演化计算。演化博弈编排的任务执行节点负载均衡,随集群负载的增加任务调度总耗时呈线性增长,计算复杂性随资源的变化改变较小。

4 结束语

本文研究演化博弈模型并设计弹性调控平台的任务调度框架,实现对大型并行计算 SCADA 系统分片应用的任务调度。由于本文调度框架涉及业务

感知的信息采集,因此下一步需将前置采集和消息总线的网络监视信息纳入计算,采用告警信息修正任务编排策略,增加测试节点数及调度任务数,实现更高效及稳定的任务调度。

参考文献

- [1] 郑宗强,翟明玉,彭晖,等. 电网调控分布式 SCADA 系统体系架构与关键技术[J]. 电力系统自动化,2017,41(5):71-77.
- [2] 梁寿愚,胡荣,周华锋,等. 基于云计算架构的新一代调度自动化系统[J]. 南方电网技术,2016,10(6):8-14.
- [3] 齐平,王福成,王必晴. 云环境下基于任务执行行为感知的可信资源调度算法[J]. 小型微型计算机系统,2018,39(4):657-663.
- [4] SALEHI M A, JAVADI B, BUYYA R. QoS and preemption aware scheduling in federated and virtualized grid computing environments[J]. Journal of Parallel and Distributed Computing,2012,72(2):231-245.
- [5] 史宝鹏,段迅,孔广黔,等. 医疗云平台资源调度策略研究[J]. 计算机工程,2017,43(8):44-48,55.
- [6] FERNÁNDEZ-CERERO D, JAKÓBIK A, GRZONKA D. Security supportive energy-aware scheduling and energy policies for cloud environments[J]. Journal of Parallel and Distributed Computing,2018,119(1):191-202.
- [7] 易侃,王汝传. 基于 Nash 均衡的网格多调度节点的任务调度算法[J]. 电子学报,2009,37(2):329-333.
- [8] 王元卓,于建业,邱雯,等. 网络群体行为的演化博弈模型与分析方法[J]. 计算机学报,2015,38(2):282-300.
- [9] 鲁亮,于炯,卞琛,等. Storm 环境下基于权重的任务调度算法[J]. 计算机应用,2018,38(3):699-706.
- [10] 鲁亮,于炯,卞琛,等. 大数据流式计算框架 Storm 的任务迁移策略[J]. 计算机研究与发展,2018,55(1):71-92.
- [11] 张晶,孙少杰,范洪博. 一种基于开销优化的高稳定性任务调度算法[J]. 计算机工程,2017,43(7):60-63,69.
- [12] 杨鹏,马志程,彭博,等. 集成 Docker 容器的 OpenStack 云平台性能研究[J]. 计算机工程,2017,43(8):26-31.
- [13] ABDULHAMID S M, LATIFF M S, IDRIS I. Tasks scheduling technique using league championship algorithm for makespan minimization in IaaS cloud [EB/OL]. [2018-05-17]. <https://arxiv.org/abs/1510.03173>.
- [14] 马晓光,刘钊远. 一种适用于 Docker Swarm 集群的调度策略和算法[J]. 计算机应用与软件,2017,34(5):283-287.
- [15] 彭丽苹,吕晓丹,蒋朝惠,等. 基于 Docker 的云资源弹性调度策略[J]. 计算机应用,2018,38(2):557-562.

编辑 陆燕菲