动态资源管理和容器技术在金融行业的架构探索和明天

Resource Scheduling & Container Technology for Financial Service

余军

Gopher China 2015

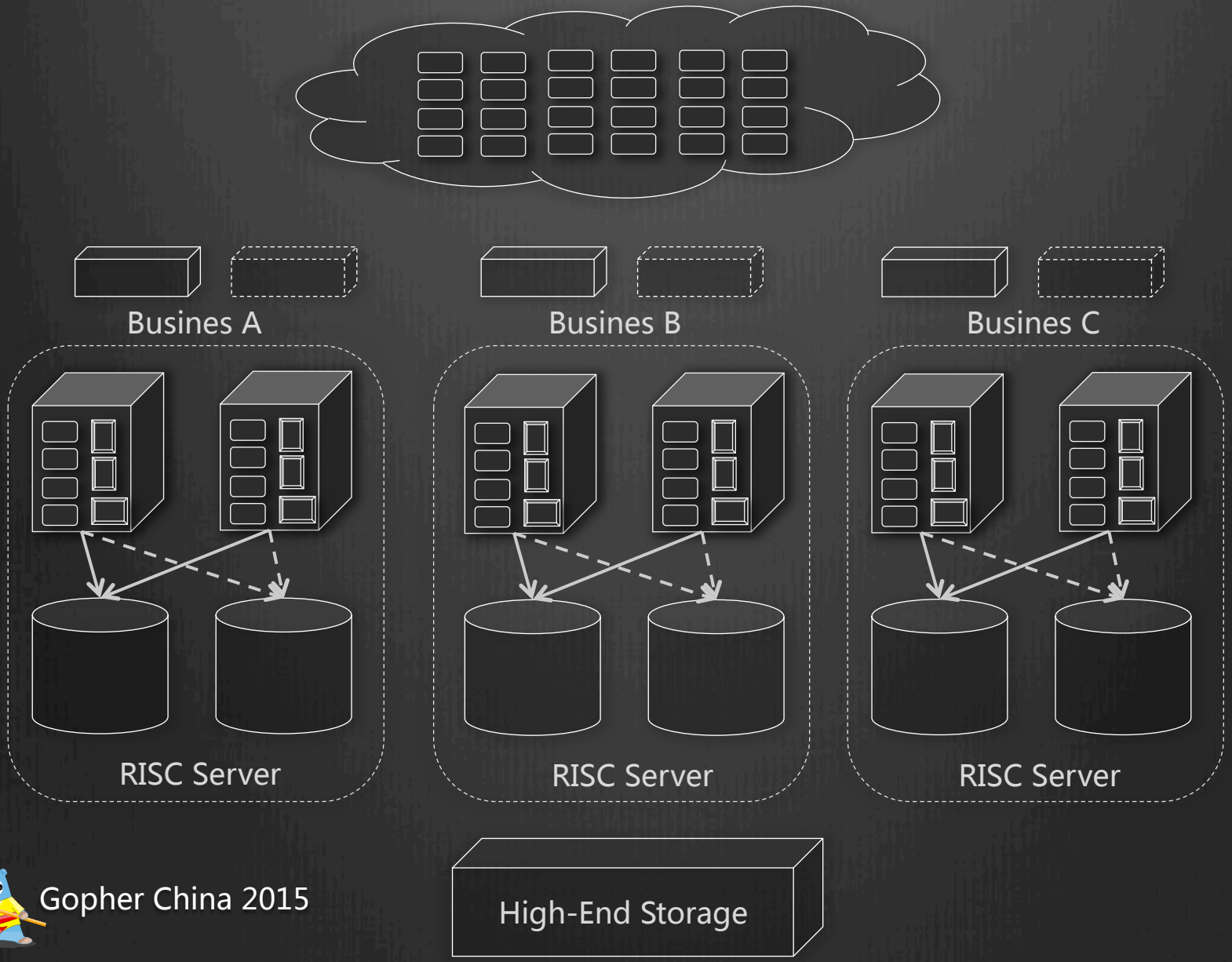
关于我

- ~19y+ Linux & opensource experience
- Co-founder of Shanghai Linux User Group (1997)
- 6y+ senior Linux and HPC engineer(Hewlett-Packard)
- 6y+ solution manager (Red Hat)
- Present : leading a fast growing opensource engineering team

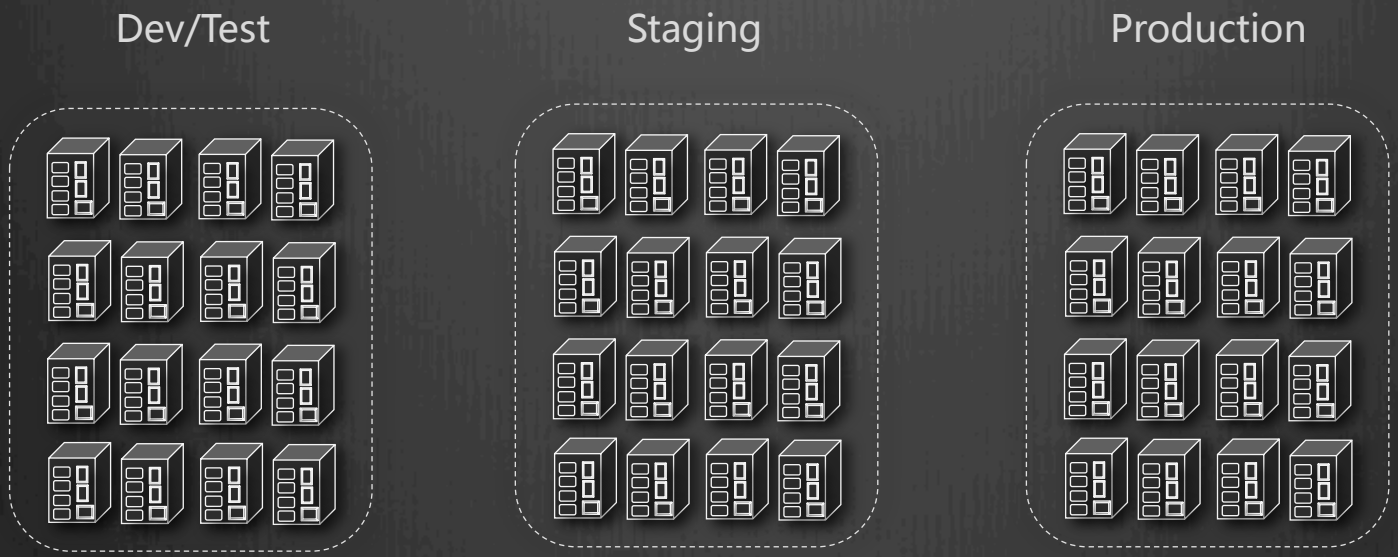
话题

- 金融行业 IT 基础架构的现状、特点、挑战
- 求解之路的探索
- 雏形和明天

金融行业IT基础架构的现状、特点、挑战



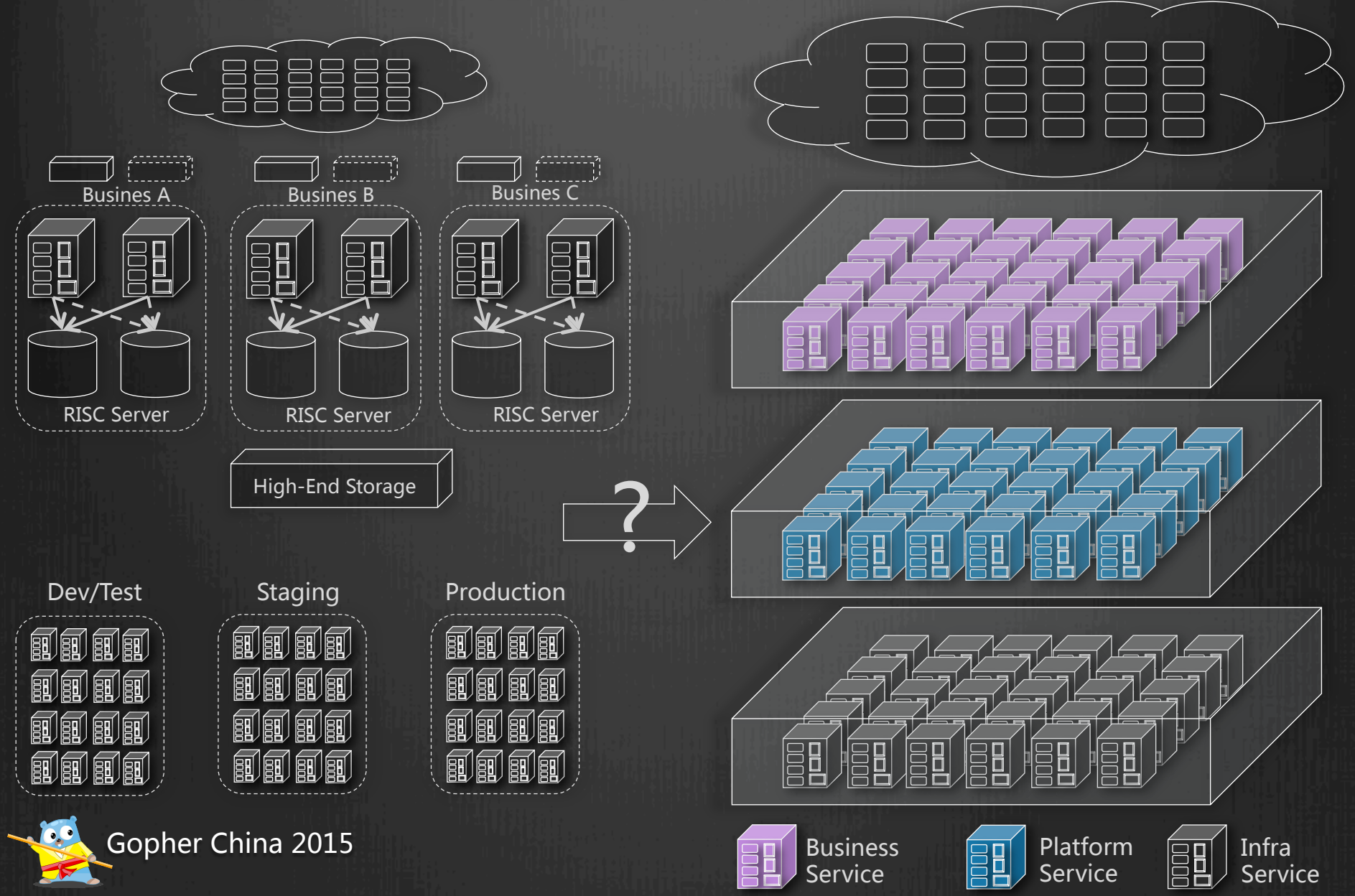
金融行业IT基础架构的现状、特点、挑战



金融行业IT基础架构的现状、特点、挑战

- “RISC to IA” 来了
- “Unix to Linux” 来了
- “开源架构”来了
- “分布式架构”来了
- “虚拟化”和“云计算”来了
- “大数据”来了
- “互联网金融”和“金融互联网”来了
- “移动业务”来了
- “敏捷”和“Devops”来了
- “去IOE”来了
- “自主可控”来了

金融行业IT基础架构的现状、特点、挑战



- 归根结底是构造一个高效的资源管控模型

求解之路的探索

- 是否已经存在相关问题域的解?
- 他们是是否解决了我们的问题?
- 我们的研究和探索

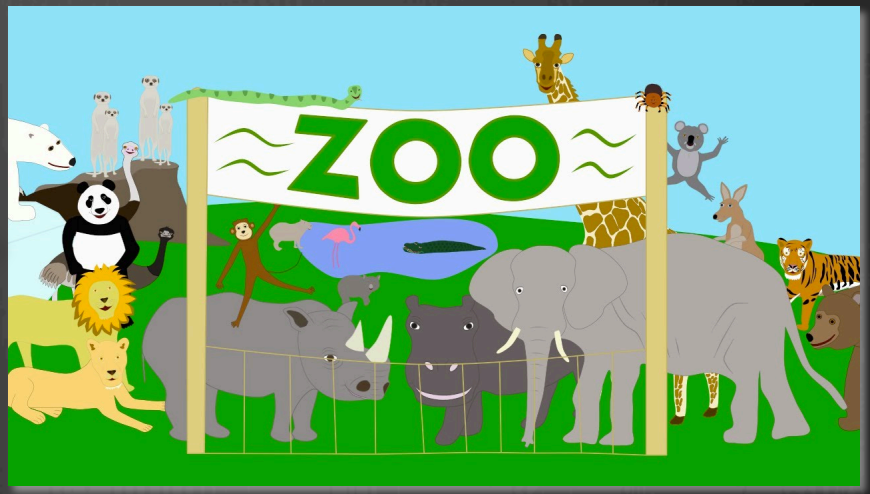
求解之路的探索

- 是否已经存在类似的解?
  - 传统: HPC中的PBS 和 Condor
  - 现代: Hadoop YARN，Apache Mesos，Google Kubernetes

求解之路的探索

- 他们是否解决了我们的问题?
  - No

the Cattle Farm and the Zoo



求解之路的探索

- 他们是否解决了我们的问题?
  - No

① Condor系统是面向高吞吐率计算计算而设计的，它的主要目的就是利用网络中工作站的空闲时间来为用户服务。

② Condor采用集中式调度模式，且不能保障用户服务质量。

③ 最小完成时间算法MCT(Minimum Completion Time)是以任意的顺序将任务映射到具有最早完成时间的主机上，它并不保证任务被指派到执行它最快的主机上，而仅关心如何最小化任务完成时间，因而可能导致任务在资源上的运行时间过长，从而潜在地增加了调度跨度。

④ Min-Min算法，利用MCT矩阵，首先分别找到能够最短完成该任务的机器及最短完成时间，然后在所有的最短完成时间中找出最小的最短完成时间对应的任务。Min-Min算法存在着一个很大的缺点，就是算法的资源负载均衡性能(Load Balancing)不高。

⑤ Max-Min算法与Min-Min算法相似，都是将任务指派给具有最小预测完成时间的主机，不同的是Max-Min算法从所有任务的最小完成时间中选取一个最大值，然后进行相应任务。主机映射，之后重复此过程直至待调度任务集合为空。

⑥ 轮询调度（Round Robin Scheduling）算法就是以轮询的方式依次将请求调度不同的服务器，即每次调度执行i = (i + 1) mod n，并选出第i台服务器。轮叫调度算法假设所有服务器处理性能均相同，不管服务器的当前连接数和响应速度。该算法相对简单，不适用于服务器组中处理性能不一的情况，而且当请求服务时间变化比较大时，轮询调度算法容易导致服务器间的负载不平衡。

求解之路的探索

- 他们是否解决了我们的问题?
  - No

Apache MESOS

Algorithm 1 DRF pseudo-code

$R = \langle r_1, \cdots, r_m \rangle$ ▷ total resource capacities
$C = \langle c_1, \cdots, c_m \rangle$ ▷ consumed resources, initially 0
$s_i \ \ (i = 1..n)$ ▷ user i's dominant shares, initially 0
$U_i = \langle u_{i,1}, \cdots, u_{i,m} \rangle \ \ (i = 1..n)$ ▷ resources given to user i, initially 0

pick user i with lowest dominant share s_i
$D_i \leftarrow$ demand of user i's next task
if $C + D_i \leq R$ **then**
  $C = C + D_i$ ▷ update consumed vector
  $U_i = U_i + D_i$ ▷ update i's allocation vector
  $s_i = \max_{j=1}^{m} \{u_{i,j}/r_j\}$
else
  return ▷ the cluster is full
end if

① Mesos 采用了DRF(Dominant Resource Fairness) 调度机制。YARN自带FIFO、Capacity Scheduler和Fair Scheduler(借鉴了Mesos的DRF)。

② Mesos中的DRF调度算法过分的追求公平，没有考虑到实际的应用需求。在实际生产线上，往往需要类似于Hadoop中Capacity Scheduler的调度机制，将所有资源分成若干个queue，每个queue分配一定量的资源，每个user有一定的资源使用上限。

③ Mesos采用了Resource Offer机制，这种调度机制面临着资源碎片问题，即：每个节点上的资源不可能全部被分配完，剩下的一点可能不足以让任何任务运行，这样，便产生了类似于操作系统中的内存碎片问题。

④ YARN适合Long running job和数据分析类资源的调度，对于数据库类等短运行时场景资源调度效果较差

⑤ YARN采用了增量资源分配机制（当应用程序申请的资源暂时无法保证时，为应用程序预留一个节点上的资源直到累计释放的空闲资源满足应用程序需求），这种机制会造成浪费，但不会出现饿死现象

⑥ Mesos 和 YARN的调度器的扩展和定制在开发上都比较繁琐。

求解之路的探索

- 他们是否解决了我们的问题?
  - No

kubernetes

① Kubernetes 仅仅是实现了一个极其简单的调度器。鼓励开发者编写自己的调度器注册进框架

② 调度策略分为两大类：Predicates和Priorities，其中Predicates判断是否将pod调度到特定minion(host)上运行，而Priorities则是在Predicates的计算基础上，通过积分Score方式，决定调度量。

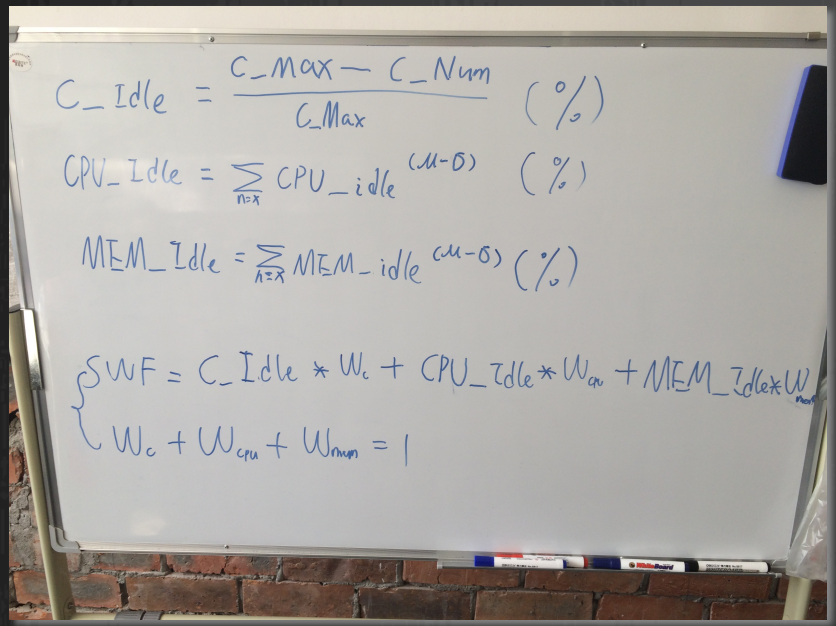
③ Predicates包括：PodFitsPorts、PodFitsResources、NoDiskConflict、MatchNodeSelector和HostName，即一个minion能够被选中的前提是需要经历前面提到的这5个Predicates的检验，而Priorities又包括：LeastRequestedPriority、ServiceSpreadingPriority和EqualPriority，分别为通过Predicates检验的minion计算优先级（score），score是一个范围是0-10的整数，0代表最低优先级，10代表最高优先级。

④ 调度机制还是过于平均，Predicates本质上作为一个过滤器，带有太多资源的物理属性。

⑤ 调度机制非常适合公有云场景，对于私有云领域欠缺灵活性。

求解之路的探索

- 我们的研究和探索 SWF – Scene Based Weighted Fairness
  - 适合金融行业架构和业务场景的资源调度机制
  - 围绕各种对资源有不同分配使用要求的应用开展调度工作
  - 针对不同生产区域(devops/test, staging, production) 实现权重调整的调度方式
  - 实现局部调度的个性化和全局调度的公平性
  - 杜绝简单粗暴的调度，对关键业务应用的运行影响降低到最小



① 基于不同应用的场景数据做资源的实时计算。

② 场景数据的短期切片和中长期切片可以适应资源池投产的不同阶段。

③ 实现了(人工)可干预的分配机制(阈值)。

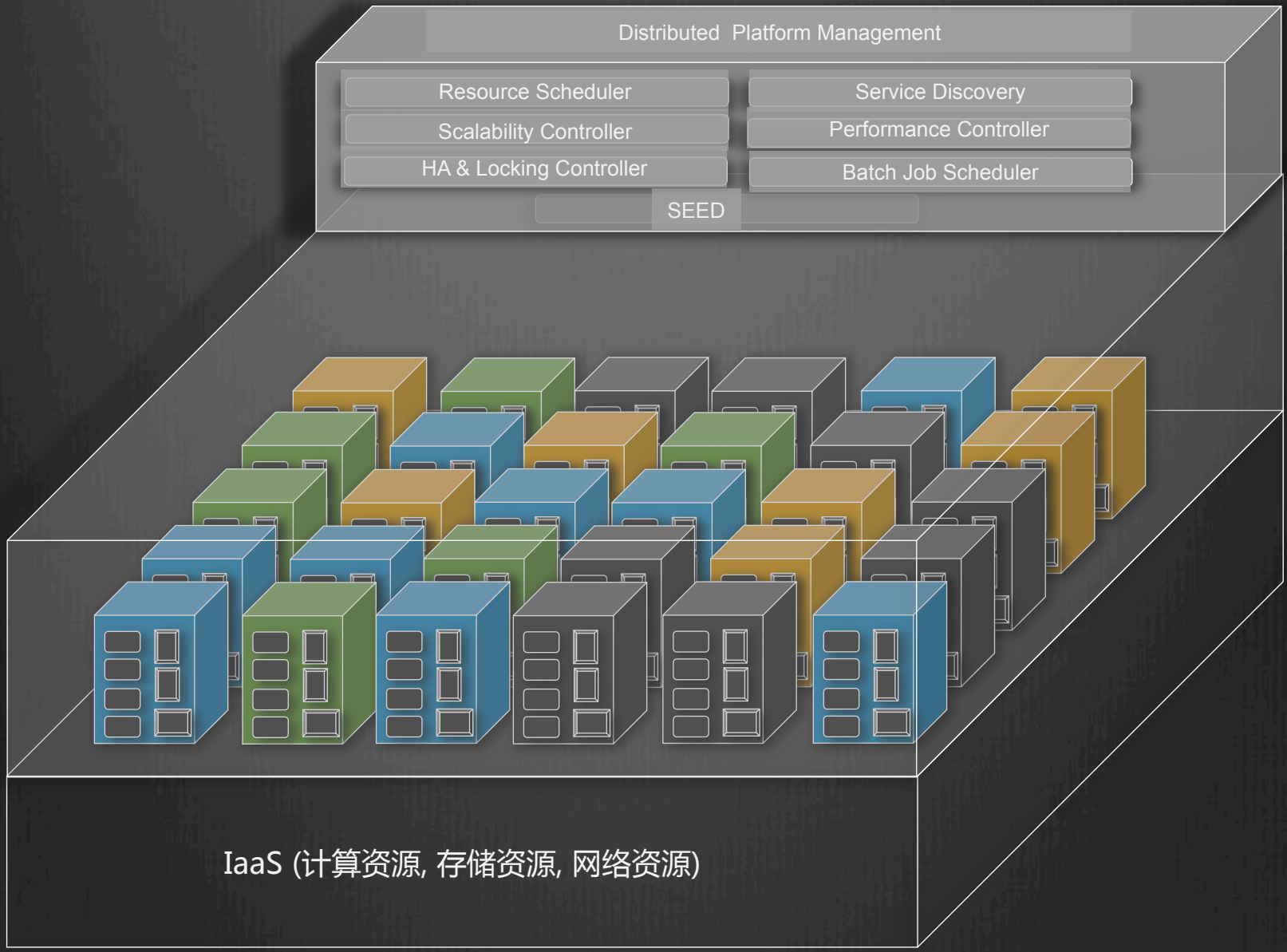
④ 通过权重比对利用率优先，容量优先和可用性优先进行调控。

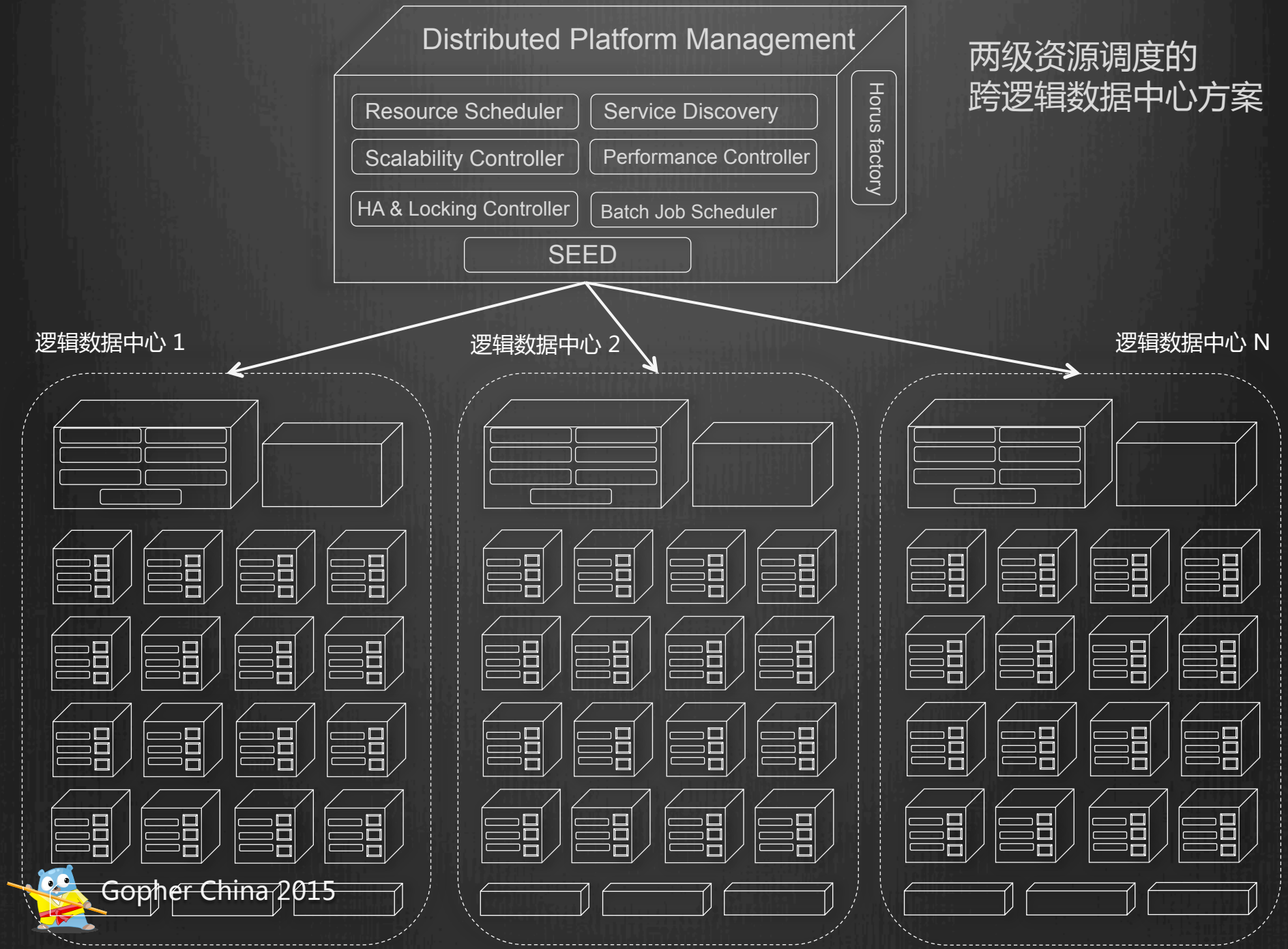
⑤ 具体实现采用较为独立的模块方式，方便将来开源后被第三方使用，定制和集成。

⑥ 面向金融行业应用场景，进行持续的演进和调整。

- 雏形和明天

- 我们在围绕Container技术正在做一些面向金融行业场景的严肃而”有趣”的东西


Distributed Platform Management
Resource Scheduler
Service Discovery
Scalability Controller
Performance Controller
HA & Locking Controller
Batch Job Scheduler
SEED
IaaS (计算资源, 存储资源, 网络资源)


Distributed Platform Management
Resource Scheduler
Service Discovery
Scalability Controller
Performance Controller
HA & Locking Controller
Batch Job Scheduler
SEED
Horus factory
两级资源调度的
跨逻辑数据中心方案
逻辑数据中心 1
逻辑数据中心 2
逻辑数据中心 N
Gopher China 2015

100% Written by Golang
Heavily leveraging Beego framework
(Thanks Asta for your great work!)
API centric desgin and development
...
Will be released under Apache License as standard opensource project

100% 自主研发。100% 开源给用户	Apache License v 2.0
容器级资源运行技术	基于Linux 内核隔离及业界先进的Container容器技术
自主研发的资源分配和动态调度调度算法	自主研发SWF核心算法 (基于场景的加权均衡算法)
两级作业调度框架	自主研发Gardener – Seed 作业调度系统
服务弹性伸缩	自主研发Lighthouse智能服务伸缩模型
分布式高可用控制系统	基于Raft/Chubby算法和GOSSIP协议的分布式高可用控制系统与服务发现
智能模板和堆叠式组件管理系统	深度优化和预置的多层堆叠式开源软件发布和管理系统，可以在保障版本统一的条件下，智能推送和维护应用模板
运维过程数据的大数据分析系统	自主研发Horus 运维数据处理平台

不喜欢因循守旧的工程师
渴望造点新东西的技术人
看到金融IT行业挑战和机遇的朋友
热爱开源的大粉丝们
我们虚位以待

http://golanghome.com/post/494

Or email : joeyu@bsgchina.com

Gopher China 2015