摘要	研究内容	总结
硬件指标采集工具，该工具补全了容器的内存子系统压力相关指标与cycles-per-instructions(CPI)的监控，前者丰富调度系统的资源感知，后者用于集群数据分析。	**4.3 系统监控** 目前主流的开源调度框架及本文公司都没法对底层共享资源压力的相关指标的采集。所以为了实现感知这类共享资源压力，本文额外开发了采集工具。本文在系统监控方面，主要依赖平台已有能力，新增了采集插件，拓展监控资源的丰富性。 **4.3.1 监控插件接入体系** 本文开发的硬件指标采集工具基于 Linux 的 perf_event_open 系统调用实现，本文所在公司采用小米公司开源的 open-falcon 产品实现监控系统，监控集群机器的各种数据与指标。 **4.3.2 硬件指标采集插件实现** 本文通过 golang 语言提供的 cgo 的方式实现硬件指标采集插件，数据输出与简单的数据由 golang 实现；而系统调用等底层方法使用 c 语言实现，即采集的核心代码逻辑部分，嵌入到 golang 程序中。	本文实现的硬件事件采集工具实现了对 CPI 的采集，该指标可作为通用的应用性能指标代理，在本文也有部分内容做了相关陈述与分析，除本文涉及的工作之外，该指标还可以用于分析应用的资源敏感性等工作。
通用的内存子系统压力定义，包括 last level cache(LLC)与内存控制器资源的压力定义。	**4.2 单机内存子系统压力定义与实现** **4.2.1 内存子系统压力的产生分析** 从计算机的内存层次结构上看，CPU 并不直接访问主存，CPU 会直接访问 LLC，需要的信息在 LLC 上找不到后，会产生 LLC Misses，然后到主存中寻找。LLC 上的信息缺失请求将发送到内存控制器上，由内存控制器处理请求。 内存子系统的压力主要包含 LLC 与内存控制器的压力。所以两者主要的压力分别来源于 CPU 对 LLC 的访问，以及 LLC 产生丢失后对内存控制器发起的请求 **4.2.2 内存子系统压力的定义** 为了评估单个节点上内存子系统的压力，通过调研本文了解到内存子系统的压力主要分为 LLC 压力与内存控制器压力，研究工作中涉及的系统指标主要有 LLC 占用量，内存带宽使用量。 本文工作的核心之一是基于比较有代表的并适用于所有架构的性能指标，提出具有通用性的内存子系统压力指标，以量化主机内存子系统压力/应用对内存子系统的需求特性。主要指标分别命名为 *Host_Cache_Pressure* 与 *Host_Memory_Controller_Pressure*，分别代表主机 LLC 与内存控制器压力。	本文根据研究工作与内存层次结构定义出可量化的压力指标，并通过集群生产环境数据与单机实验验证这些指标的有效性，这对 Kubernetes 在调度阶段优化集群内存子系统资源压力有重要的价值，并且因为指标具有通用性，所以本文的量化指标在集群调度方向都适用。

应用资源画像与基于画像的调度策略，本文采用通用且易生产落地的方法构建了应用资源画像，并且基于画像设计了基于高斯估计的调度优化策略，为集群跨节点资源负载(cpu/cache/memory controller)的均衡度提升最高达(38%,28%,32%)。	**4.4 调度优化设计** **4.4.1 资源负载感知的调度模块结构** 本文的调度框架实现基于公司内部 Kubernetes 发行版，为本文的工作免去了大量冗余复杂的工作量。在开源的 Kubernetes 的基础上，多了一些核心组件，整体参与调度的核心部分为应用实例(Pod)，画像缓存(Portrait Cache)，预选策略(Predicates)，优选策略(Priorities)，节点(Node)，应用资源画像数据库（Portrait Database）。 **4.4.2 调度流程中重要数据结构及方法描述** **4.4.2.1 应用信息数据库(Portrait Database)** 存储“资源画像”部分工作对应用历史数据构建的画像数据，数据库由公司内部提供，本文主要新增了应用资源画像表。 **4.4.2.2 应用实例(Pod)** 应用实例的定义在前文“相关技术”已阐述过，Pod 是应用的实际工作负载实例，Kubernetes 的最小调度单元。如表 4.5 所示为本文主要使用的 Pod 关键信息，Requests 为 Pod 中各容器的资源需求，此处简写，在调度时，会通过工具类将所有容器全部累加参与调度。 **4.4.2.3 节点(NodeInfo)** K8S 中节点由 Node 数据结构描述，在调度时会将 Node 封装成 NodeInfo，Kube-Scheduler 通过 NodeInformer 的 List-Watch 机制与对象缓存了保证 Lister 可以实时返回集群的节点情况。 **4.4.2.4 画像缓存(Portrait Cache)** 画像缓存体现在本文于 Kube-Scheduler 中新增的一个数据结构。画像缓存主要存储应用与节点的资源画像数据，节点的资源画像由其调度到该节点的 Pod 资源画像数据组成。 **4.5 应用资源画像实现** 本文的应用资源画像主要是:（1）应用资源容量评估，包括 CPU，内存两种资源。（2）应用资源消耗特征，包括 cpu cores，cache-misses，crpki。 **4.5.1 应用资源画像工作逻辑** **4.5.2 应用资源消耗特征分析** **4.5.3 应用资源画像算法** **4.6 基于资源画像的调度策略实现** **4.6.1 资源负载感知的策略实现**	本文设计了基于应用资源画像的调度策略，分别是基于容量推荐的预选优化策略，与基于高斯百分位估计的优选优化策略，其中，依赖的应用资源画像都主要采用概率统计的方法，这类方法在生产环境中易于落地，且通过实践验证，也能有不错的效果，另外，本文采用的应用资源画像与调度策略可以容易地拓展到其他资源，如网络 I/O，硬盘读写的 IOPS 等。

调度仿真工具，实现对Kubernetes组件的端到端的调度验证。	**4.7 调度仿真** 本文参与了基于开源工具 Virtual Kubelet[52]开发的集群仿真工具，本文将通过使用 Virtual Kubelet(VK)框架实现对 Kubernetes 节点的模拟，避免需要真实工作节点资源。 **4.7.1 调度仿真设计** 集群调度仿真部分涉及待测试的 Kubernetes 主集群，模拟工作节点（Simlet），调度序列读取工具（Cluster Recorder），调度序列输入工具（Cluster Stream），调度仿真可视化（monitor/grafana/telegraf/influxdb），环境部署处理服务（Deploy Server）。 **4.7.2 调度仿真实现** **4.7.2.1 工作节点伪装** **4.7.2.2 模拟环境部署** **4.7.2.3 调度序列生成与输入**	本文考虑到调度仿真在Kubernetes生态的缺失，设计与实现了对Kubernetes组件运行态端到端的仿真方案，使得调度团队可以在新的调度方案上线前，进行比较完备的调度仿真，避免潜在的风险。