南京邮电大学

毕业设计(论文)外文资料翻译

|  |  |
| --- | --- |
| 学 院 | 物联网学院 |
| 专　　业 | 网络工程 |
| 学生姓名 | 徐阳 |
| 班级学号 | B13070134 |
| 外文出处 | International Conference on. IEEE |

|  |
| --- |
| 指导教师评价：  1．翻译内容与课题的结合度： □ 优 □ 良 □ 中 □ 差  2．翻译内容的准确、流畅： □ 优 □ 良 □ 中 □ 差  3．专业词汇翻译的准确性： □ 优 □ 良 □ 中 □ 差  4．翻译字符数是否符合规定要求：□ 符合 □ 不符合  　　　　　　　　　　　　　　　　　　　　　　　指导教师签名：老师签名  2017 年　5　月　19　日 |

**Kestrel：基于XMPP框架的多任务计算应用程序**

Lance Stout，Michael A. Murphy和Sebastien Goasguen计算学院

克莱姆森大学

Clemson，SC 29634-0974 USA

{lstout，mamurph，[sebgoa}@clemson.edu](mailto:sebgoa%7D@clemson.edu)

**摘要**

本文介绍了基于可扩展消息传递和存在协议（XMPP）的多任务计算（MTC）应用程序的新的分布式计算框架。已经开发了一个名为Kestrel的轻量级高可用性系统，用于探索基于XMPP的技术，以改进MTC系统对扩展和间歇性计算代理导致故障的容错事件。通过在大型即时通讯系统中使用可扩展到数百万客户端的杠杆化技术，该MTC系统旨在以不同的粒度级别扩展到数百万个代理：内核，机器，集群甚至传感器，使其成为一个优秀的MTC。

Kestrel的架构灵感来自于网格上的试点工作框架以及僵尸网络的分布式设计，同时增加了用于通信的消息即时通讯协议。而僵尸网络命令和控制系统经常使用网络中继聊天（IRC），分布式哈希表（DHT）和其他对等（P2P）技术的组合，Kestrel利用XMPP的存在通知功能，使得系统能够实时地维持机器存在状态的连续跟踪。 XMPP也可以轻松扩展到特定于应用程序的子协议，可用于传输机器配置文件描述和作业需求。 这些子协议可以使用类似于Condor高吞吐量计算（HTC）系统中的ClassAd的机制来实现作业到系统的分布式匹配。

**关键词**

MTC, XMPP, grid(网格), distributed(分散式), cluster(簇), scheduling(调度)

**1.介绍**

多任务计算（MTC）应用程序涵盖了广泛的配置，但是在短时间内利用“大量计算资源来完成多计算任务，其中主要度量是以秒为单位”是一个的重要方面。在可以根据需求创建和销毁计算节点的应用程序中，所需的时间粒度上编组和释放计算资源在网格和虚拟组织集（VOC）等重叠中是有问题的。

Kestrel旨在探索为异构，间歇连接的计算节点创建容错，可扩展和跨平台作业调度系统的方法。Kestrel通过建立在可扩展消息传递和存在协议（XMPP）上，解决了计算代理和用户代理中断连接所造成的问题。XMPP提供的存在通知系统，允许Kestrel管理器节点实时跟踪计算节点池中的资源可用性。 Kestrel可以立即从可用节点列表中删除计算节点，从而阻止在不存在的计算机上调度作业。其他间歇连接的代理不一定具有计算性质。例如，具有网络功能的一小群机器人代理可以加入到资源池中，其中调度作业是要执行的物理动作。当前即时消息系统的规模表明，基于XMPP的框架将允许创建用于实时计算的超大异构代理池。跨平台标准是通过使用SleekXMPP库在Python中来实现的。

本文的其余部分安排如下。 相关工作在第2节中介绍。第3节讨论了Kestrel的网络架构，之后在第4节介绍了特定的网络协议。第5节描述了用于传输Kestrel消息的消息协议格式。第6节介绍了Kestrel当前实施的架构概述。第7节介绍了结论和未来工作。

**2. 相关工作**

Condor高吞吐量计算（HTC）系统

专门管理计算密集型工作，并在全球范围内用于管理网格和集群。Condor架构由运行收集器和谈判者守护进程的中央管理器组成，提交运行schedd守护进程的节点，并执行startd守护进程的节点。提交和执行节点与中央管理器通信，以共享描述机器功能的ClassAd。作业请求与Clas-sAds中公布的功能定期地相匹配。一旦进行匹配并且受影响的节点通知，则在匹配的提交和执行节点之间建立直接连接，以允许作业可执行文件和作业输出在机器之间传送。虽然Condor是物理主机上的循环清理和集群计算的优秀系统，但提交主机和工作人员之间的直接连接不能跨NAT绑定。此外，由于收集器更新周期性不是连续发生，所以在具有间歇连接的工作节点的环境中，操作可能是次优的。 Kestrel的灵感来自于Condor架构，但替代了与XMPP的所有通信，增加了实时计算能力以及极大的规模。

在Weis和Lewis中实施了基于XMPP的调度系统，用于特殊网格计算。该系统专门用于并行优化的曲折线射频识别（RFID）天线的计算。通过调度程序可以使用的静态机器列表，控制的XMPP客户端将联系任何未连接到XMPP服务器的计算机，以启动XMPP工作守护程序。虽然结构类似于Kestrel，但调度程序旨在将单一类型的作业分配给已知的机器; 它并不意味作为一般的MTC框架。

Wang等提出的一种混合对等（P2P）僵尸网络，通过一个由P2P协议互连的仆人机器人组来代替典型僵尸网络体系结构中发现的单一命令和控制机器。作为避免检测的另外一种手段，僵尸网络不会使用其他僵尸网络通常使用的互联网中继聊天（IRC）来进行通信。Kestrel利用类似的分布式命令和控制架构来增加容错能力; 确实，多个XMPP服务器提高了冗余和水平的可扩展性。

**3. 网络架构**

Kestrel池中的网络由四种类型的节点组成：管理员，工作人员，用户和XMPP服务器。单个物理机或虚拟机可以存在多个节点。XMPP服务器可能被配置为使用分布式数据库充当集群，以保持名单和状态信息的同步。然后，所有其他节点连接到XMPP服务器。 由于中间人用于通信而不是要求节点直接相互连接，因此NAT穿越不是问题。

用户节点是用于提交作业请求或查询先前提交的作业的状态的客户端应用程序。 这些节点是高度瞬态的，仅在将用户请求发送到管理器节点并等待回复时才连接。 这样就可以执行作业，而不要求提交者的计算机保持连接到池。在用户下次连接系统后，可以为先前提交的作业请求完成通知。

管理器节点作为Condor池中收集器和协商者组合的等价物。因此，在Kestrel池中发送的每个消息都被发送到管理器节点或从管理器节点发送。由于管理员处理的大量消息，特别是存在通知，有两类管理员使用两种不同的XMPP实现方法来解决。第一类管理器是一个常规的XMPP客户端应用程序，对于只有几百到一千个节点的池有用。通过使用常规客户端，如果管理员断开连接，则XMPP服务器可以立即向工作人员和用户节点发送通知。然而，一旦一个游泳池已经超过数千台机器，启动时间会大大增加，因为这个名册必须发送给管理员，阻止任何其他消息。因此，第二类管理器是一个XMPP服务器组件，它能像一个正常的客户端一样运行，除了它维护自己的名册而不是依赖于服务器。如果任何一种类型的管理员都会自动执行，则池的其余部分仍然可以完成任何以前计划的作业。

工作节点跟踪本地机器状态并处理实际的作业执行。使用XMPP存在通知，工人通过改变其在线状态为“可用”时，可用于运行排队的作业来提醒管理者。一旦作业开始，工作节点状态更改为“忙碌”，允许管理者安排新的工作岗位时绕过所要求的工人。倘若一个工人经过NAT或底层机器断电，经理会收到一个线下通知，允许已在失败的工人运行的任何作业要在另一个工作改期。

**4. 网络协议**

**4.1标识符**

XMPP系统中的每个实体都有一个唯一的标识符，称为Jabber ID，或JID。这些标识符的格式为username @ server / resource。 JID或裸机的用户名和服务器部分与普通电子邮件地址的结构相似。 如果指定，可选资源字符串允许一个身份与XMPP服务器有多个连接。 在Kestrel中，通过为每个物理机分配用户名，组织工作代理池，允许主机由裸机识别。 通过向JID添加资源来单独寻址机器的每个CPU内核。 因此，四核机器中的第三个核心可以由worker362 @ kestrelpool / 3识别。 通过允许池中的每个代理拥有自己的密码来提供安全性。

一个替代实现将是使用单个裸JID来解决池中的所有工作节点，每个工作者可以通过添加资源标识符来唯一寻址，从而产生JID，如worker @ kestrelpool / 362。 通过使用不同的用户名，工作人员可以通过物理位置，操作系统或预期用途在逻辑上分门别类。 使用此命名约定有利于大大减少管理员代理人名册中的条目数量。 但是，此实现需要共享同一裸机的所有计算机共享相同的密码。

Kestrel能够利用命名约定。 此外，只要池中的每个代理具有唯一的JID，Kestrel可以扩展到其他命名约定。

**4.2消息类型**

XMPP协议提供三种不同的消息类型。第一种类型，存在通知用于指示代理的可用性，并且可以携带描述代理当前状态的状态消息。聊天消息主要用于在人之间发送即时消息。 这些消息不需要任何回复确认。 XMPP保证聊天消息将按顺序接收，但不会让他们到达。 需要响应的消息称为“iq”请求。 这些请求返回状态码以响应每个查询消息。 在iq消息中，可以传输遵守ad-hoc命令格式或其他自定义命名空间的XML内容的所有内容。

Kestrel使用存在通知来跟踪工作者代理和聊天消息的可用性，以便在代理之间通信命令和数据。其他基于XMPP的调度系统已经使用了用于代理间消息的XML协议的iq消息方法。这种安排由Weis和Lewis描述，用于生成RFID天线的自组网格计算系统。然而，为了简化开发和测试，Kestrel使用可以通过即时消息客户端发送的聊天消息。因此，可以使用Pidgin，Adium或任何其他支持XMPP的即时消息客户端以及通过正常的命令行程序来管理池。聊天消息的内容遵循使用JavaScript对象名（JSON）而不是XML的自定义子协议。内容存储在一个JSON字典中，它总是包含一个指定消息目的的“类型”属性。例如，一个worker\_profile类型将指示包含从工作代理发送到管理器代理的机器属性的消息，如图1所示。在图2中显示了在工作人员的生命周期中生成的典型消息序列。

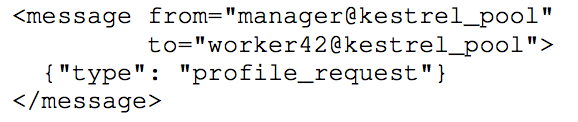


图1：有关由工作代理管理的机器的详细信息的请求。

**5. 简介格式**

用于表示机器配置文件的格式（Condor ClassAd的Kestrel等价物），基于JavaScript对象符号（JSON）。JSON标准以词典的形式提供键值对，并且还支持列表和其他数据结构。通过使用此格式，可以使用标准库调用轻松地读取和解析机器配置文件。几个机器属性是通用的，例如操作系统的名称，CPU内核的数量和可用RAM的数量。另外，可以通过使用标签来指定机器的选项属性。每个机器配置文件可能包含一个“提供”部分，由图3所示的自定义标签列表组成。

作业请求的格式类似于机器配置文件。 区别在于它指定了一个工作需要什么，而不是一个机器提供的。 JSON标准没有用于表达需求规范所需的关系的内置工具。

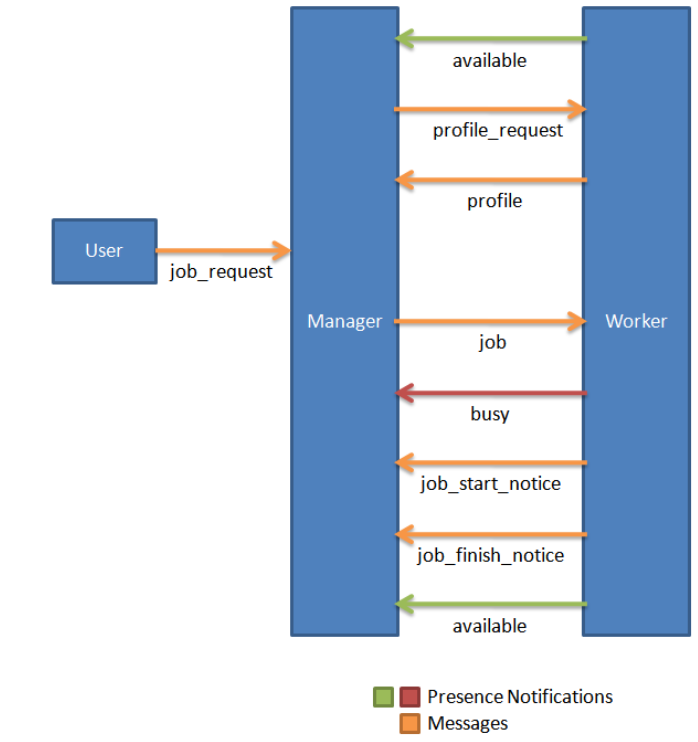


图2：工人代理生命周期内的信息流。

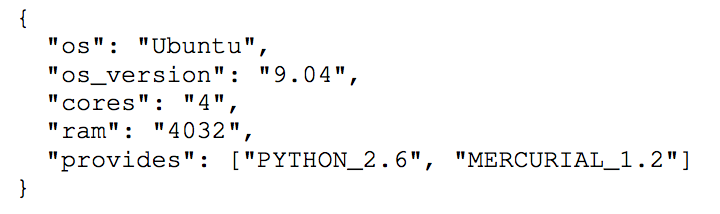


图3：样品机配置文件

为了克服这个限制，开发了一种使用规范和列表来表达关系的方法。 每个作业请求都包含可以是字典或列表的属性“require”。 在列表的情况下，它只是一个必须匹配的自定义标签的列表。 词典表示要求更具体的规则。 字典中的每个键都是在关系中比较的属性的名称。 通过为该属性分配一个字符串或数字值来表示等式。 因此，为了表示“核心== 2”，将使用以下内容：

"requires": {"cores": 2}

然而，有时候可能会有一组可能的值。 在这种情况下，将一个值列表分配给该属性以表示OR关系。 关系“cores == 2 OR cores == 4”将表示为：

"requires": {"cores": [2, 4]}

另一个用例是指定除了相等之外的关系，比如大于关系。 此外，由于指定多个这样的关系通常在AND表达式中有意义，所以使用字典来表示这些要求。 对于“core> 1和core <10”的关系，则表示形式为：

"requires": {"cores": {">": 1, "<": 10}}

列表和词典语义可以组合起来，形成更强大的表达。 指定一个作业需要一个“core = 4 OR cores> 6”的机器，那么“requires”属性就是：

"requires": {"cores": [4, {">": 6}]}

同样，语句“cores！= 2 AND cores！= 5”可以表示为：

"requires": {"cores": {"!=": [2, 5]}}

任何需要复杂关系规则和匹配一组标签的作业请求，可以在“需要”部分中使用“has”属性。

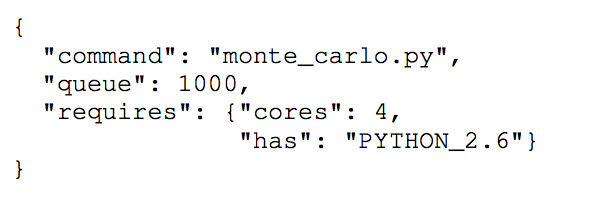


图4：在安装了Python 2.6的四核机器上执行monte carlo.py 1000次的作业请求。

**6. 程序式架构**

Kestrel软件具有多个逻辑组件，包括XMPP接口，数据库接口和作业调度机制。 早期版本的Kestrel将这些组件组合成一个具有所有系统逻辑的单一层在XMPP客户端代码中。早期实施的单元测试显示与紧耦合有关的问题。 因此，实施架构被改变为事件驱动设计。

事件驱动的架构允许程序的一部分彼此分离，导致松散的连接。可以建立去耦边界，其中边界内的所有组件可以直接相互访问，并且可以通过事件将消息发送到解耦网络。在Kestrel中，这些事件由中心内核管理。事件名称在应用程序启动时向对应的事件处理程序注册。 注册允许使用单个代码库通过选择特定于角色的事件来创建工作人员，管理员和用户的不同程序变体。

图5显示了各种事件处理程序集合之间的关系。 这些集合可以在三个程序变体中的任何一个内部使用，也可以用作外部世界和内部系统之间的接口。 对于生产使用，XMPP事件处理程序提供了接口组件。 为了测试目的，单元测试运行器提供了一个单独的，更有限的界面。

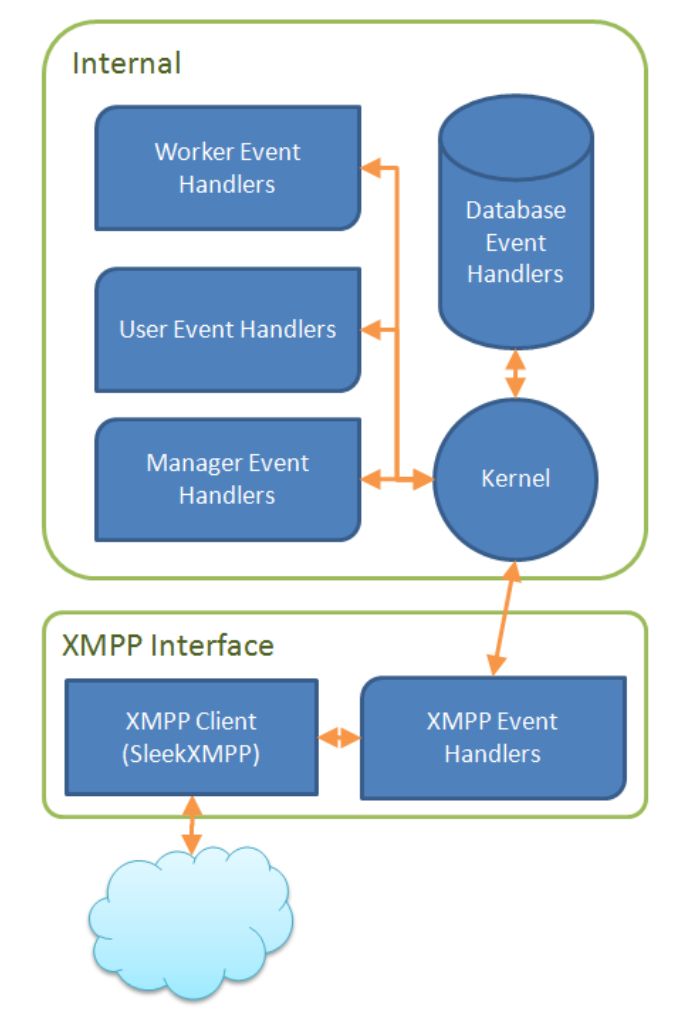


图5：程序序架构。

**6.1事件类型**

Kestrel有三种主要的事件类型。 第一组与数据存储有关。 处理XMPP事件时，不要调用数据库函数或将SQL直接传递到数据库，而是为每种类型的实体创建，读取，更新或删除（CRUD）操作。例如，当工作人员上线时，会触发update\_worker\_entry事件，从而更新其在数据库中的在线状态。对于读取操作，数据库事件处理程序也将返回相应的数据库记录。

第二组事件可以分为与三个程序变体相匹配的子组。工作人员事件与接收作业请求以及启动和停止作业执行有关。 用户事件是最简单的组，接收关于工作状态和处理任何取消请求的向管理员查询的响应。

最大的事件集合是管理员相关的。 除了由工作人员状态变化触发的事件用户请求，还有作业调度事件。 每个作业请求和取消后触发schedule\_jobs事件。每当工人变得可用时。 这个事件是不定期，因为不能安排的工作，在池的状态发生变化之前，仍然不能运行。 由于调度是系统中的单独事件，调度算法可以通过指定来轻松替换注册期间的不同事件处理程序。

最后一组事件与节点通信有关。在Kestrel的当前实现中，这些事件由构成内部系统和网络之间接口的XMPP特定功能处理。特别地，在许多情况下，必须在节点之间发送消息，例如通知工作人员工作匹配。在那种情况下，触发一个send\_job\_request事件来提醒XMPP接口使用事件中提供的数据发送实际消息。只要可以提供同一组事件，就可以使用其他网络协议创建其他接口。

**6.2事件解剖学**

每个事件都带有两条信息：事件触发期间传递的数据和事件跟踪（图6）。 当程序的顶层触发事件时，事件跟踪为空。 当事件处理程序完成处理并返回顶层时，跟踪将包含由于初始事件而触发的每个事件的名称。 除了每个事件的名称之外，跟踪还包括传递给该事件的数据。 对于不在顶层发生的事件，跟踪将包含已被触发的事件的信息。 跟踪是重要的，因为它允许使用单元测试来检查事件链。

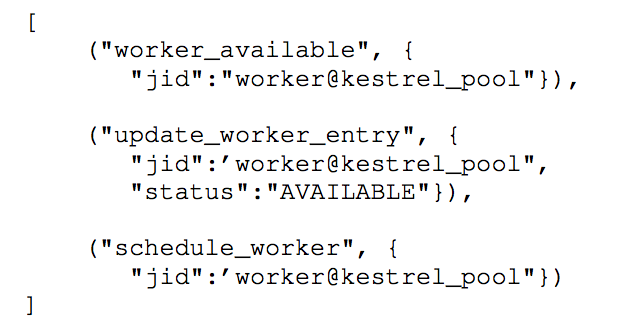


图6：由工作者生成的事件跟踪将其状态更改为可用。

**7. 结论和未来的工作**

使用XMPP在计算节点之间进行通信可实时实现资源跟踪。 通过消除计算节点断开连接和管理节点接收到断开连接的通知之间的延迟，可以为服务器安排较少的作业。实时跟踪也适用于管理器节点，允许系统立即生成新的管理员节点来应对任何中断或重载，从而保持容错和可扩展性。

Kestrel的未来扩展可能包括更有力的机制来减轻节点断开后潜在的数据丢失。例如，当管理员离线时，作业队列数据必须保持不变。池中的其他信息本质上是分散的：每个工作人员都知道运行它的本地机器的细节。因此，随着工作人员重新上线，系统恢复了机器状态信息。但是，作业信息集中在管理器节点中。在终止或中断的情况下，如果管理员被替换为在池中没有任何工作知识的新节点，那么所有挂起的作业都将丢失。在更为理想的情况下，同一管理员节点能够以其完整的数据库重新启动，从而避免了在中断期间提交的作业之外的作业丢失。

为了确保在失去经营者之后保留工作，可以使用简单的档案策略。除了执行作业之外，工作者节点也被用作存档器，可以将作业请求保存为备份。在接收到工作请求后，管理员选择一些工作人员，然后转发作业进行存储而不是执行。当工作节点上线时，发送给管理员的配置文件包括一个标志，指示数据是否已经存档在工作人员身上。 管理员可以请求发送归档数据，以便可以使用工作者提供的任何新作业条目更新数据库。 不同的存档策略是可能的，可以选择不同的工作组来存储数据库的部分。