# Partisan: Scaling the Distributed Actor Runtime

学院：计算机科学与技术学院 学号：201973263 姓名：吴泽

一 论文背景

如今，对于应用程序开发人员而言，因为并发、状态管理、并行所带来的挑战，构建分布式应用程序仍然是一项艰巨的任务。而Actor编程模型通过封装状态、允许受控以及串行访问的特性非常好的解决了分布式系统所带来的挑战，因此在许多工业领域得到了广泛采用并且实现大规模的成功。现有三个主要的工业级分布式actor系统。分布式Erlang，Akka （适用于Scala）和Microsoft的Orleans （适用于C＃）。

然而，这些分布式参与者系统在可扩展性和延迟方面仍然受到限制

可扩展性：与可以支持数百到数千个节点的其他分布式框架相比，这些生产级的分布式actor系统可以支持的节点数量受到限制。例如，分布式Erlang尚未在大于200个节点的集群上运行，而基于分布式Erlang构建的较流行的应用程序之一，即分布式数据库Riak，已证明不能扩展到超过60个节点。

延迟：由于其基本的计算模型（即在具有独立队列的角色之间在角色之间传递的单向异步消息被复用到节点之间的单个队列中），分布式Actor系统经常遭受排头阻塞的问题。 例如，分布式数据库Riak避免使用分布式Erlang进行后台数据同步，以避免在读/写请求路径中出现行头阻塞。

为了解决这些限制，本文提出了一种替代运行时系统的设计，以提高分布式Actor系统的可伸缩性和性能，以及该运行时的一种称为PARTISAN的实现。 PARTISAN通过允许应用程序开发人员将覆盖网络专用于应用程序在运行时所需的通信模式，从而实现了更大的可伸缩性，而无需更改应用程序的语义。 PARTISAN通过为应用程序开发人员提供三种自定义消息传递行为的方式来降低延迟，而无需更改应用程序的语义或更改应用程序代码。 PARTISAN使应用程序开发人员能够（i）自定义并行性（用于增加节点之间的通信通道的数量），（ii）利用命名通道（用于分隔参与者之间发送的不同类型的消息），以及（iii）进行亲缘化调度（用于根据消息源，目标和类型在通信通道之间划分流量）。

二 提出方法

为了解决网络覆盖引起的问题，PARTISAN支持在运行时选择指定的网络覆盖。 PARTISAN的API公开了网络覆盖层不可知的编程模型——仅异步消息传递和集群成员身份操作——可以轻松地使程序员构建可以在任何受支持的网络覆盖上运行的应用程序。 在运行时选择特定的网络覆盖仅会影响应用程序的性能，而不会更改应用程序的语义。 网络覆盖的选择是在运行时指定的配置参数完成的。 因此，更改覆盖网络不需要重新编译，并且在应用程序的生命周期内选择是固定的。

PARTISAN支持四个覆盖，并为开发人员提供了一个API，供开发人员使用自己的网络覆盖来扩展系统：static, full-mesh, client-server, and peer-to-peer.

**·static, full-mesh, client-server overlays**

这三者是类似的，每种覆盖使用单个连接在群集中每个节点之间进行通信。通过监视此连接来执行故障检测。断开此连接后，该节点将报告为关闭。使用静态覆盖，成员资格在运行时是固定的，而使用全网格覆盖，成员资格是动态的，可以在系统运行时进行更改。使用客户端-服务器覆盖，类似于传统的中心辐射型拓扑，仅在服务器之间以及从服务器到客户端之间保持连接。

**·peer to peer overlays**

对点覆盖建立在HyParView 成员协议和Plumtree 广播协议的基础上，它们都使用两阶段方法配对有效的传播具有弹性修复协议的安全协议，用于确保网络分区期间的操作。并且每一个peer只维护局部视图。对于局部视图，节点可能希望向未直接连接的其他节点发送消息。为了维持现有角色系统的现有语义，PARTISAN需要支持集群中任何两个节点之间的消息传递。为了实现这一目标，PARSATISAN的对等成员后端使用Plumtree协议的实例来计算植根于每个节点的生成树。当发送到未直接连接的节点时，生成树将以尽力而为的方法沿树的叶子向下转发消息，以将消息传递到所需的节点。

在分布式环境下，Actor系统中维护集群运行的网络流量方面可能都会带来很大的网络开销，并且通过将特定于应用程序的消息传递延迟到集群维护消息传递之后，会导致延迟加剧，除了这些后台流量外，一种类型的特定于应用程序的消息传递也可能会延迟不同类型的特定于特定应用程序的消息传递，例如在慢速发送者被任意延迟在快速发送者之后的情况下。这些都是行头阻塞的所有特定情况。为了减轻这些问题，我们为应用程序开发员提供了三种方法来自定义分布式参与者系统中的消息传递行为。通过(i) customizing parallelism, (ii) utilizing named channels, and (iii) affifinitized scheduling.

**·Parallelism**

为了减少单个消息队列的排头阻塞的影响，可以引入其他消息队列以尝试并行化尽可能多的工作。我们将此机制称为并行性。在应用程序开发人员很少输入的情况下（仅指定每个目标节点在每个节点上要运行的队列数），系统可以使用随机或循环调度将工作分配给队列。在大多数情况下，系统可以根据可用的系统资源来最佳选择此参数。

**·Named Channels**

尽管并行性可以增加并行执行的工作量，但是如果这是唯一用于减少延迟的技术，则后台消息可能会在特定于应用程序的消息前面排队，从而导致收益递减。如果我们将这些消息队列进一步分类为后台消息队列或特定于应用程序的消息队列，则可以使我们的调度更加智能。这可以使用命名通道来实现，并且类似于许多现代网络系统中存在的服务质量（QoS）。该机制仅要求应用程序开发人员注释正在发送的消息类型，并且基于类型的专用队列用于调度这些消息。此机制允许系统自动将后台消息传递放置在不会干扰特定于应用程序的消息传递的队列中。

**· Affifinity**

尽管命名通道可以防止后台消息直接干扰特定于应用程序的消息，但是特定应用程序的消息传递仍可能受到以不同速率发送的参与者之间的干扰。 在有多个传出队列可用（并行性）的假设下，随机或循环调度仍可能产生导致排头阻塞问题的调度。 知道参与者有（i）独特的身份（指向每个参与者并且可以自己交换的唯一引用），（ii）并按顺序行动，我们可以通过选择一个来进一步优化消息调度算法。 基于发送方的身份的传出消息队列。

而本文中提出的PARTISAN是一个运行时系统，可为分布式actor应用程序提供更大的可扩展性并减少延迟。 PARTISAN通过允许应用程序开发人员将覆盖网络专用于应用程序的通信模式来提高可伸缩性。 PARTISAN通过利用几个主要的自动优化功能（可实现消息的有效调度）来降低延迟。 PARTISAN是第一个向应用程序开发人员公开此级别控制的分布式actor系统，从而提高了现有actor应用程序的性能并启用了新型actor应用程序

三 实验与评估

为了评估PARTISAN，本文设计了一组实验来回答以下问题

•RQ1：在多个并行TCP连接上使用actor消息化有什么好处？

•RQ2：在运行时选择网络覆盖是否可以为应用程序提供更好的扩展性？

针对第一个问题，此实验使用具有16个vCPU和64 GB内存的单个Linux虚拟机。在这台机器上，该实验运行了两个Erlang VM（使用PARTISAN中的API来实现）实例，这些实例使用TCP相互通信，其模拟RTT延迟为1毫秒（单个AWS可用性区域内的RTT）或20毫秒（同一AWS区域中两个可用性区域之间的RTT）。 ）单个Linux VM用于托管Erlang VM的两个实例，以确保不受外部网络的干扰并在实验期间确保固定的延迟。故意使该虚拟机保持低负载，以免看到Linux VM内部资源争用对延迟的影响。每个Erlang VM配置为在启用内核轮询的情况下运行16个调度程序。每个微基准在不断增加的延迟和有效负载大小下运行PARTISAN的多种配置，每个参与者每个实验固定的10,000条消息数量。我们考虑禁用并行性的PARTISAN，具有并行性的PARTISAN和具有固定并行性的PARTISAN。

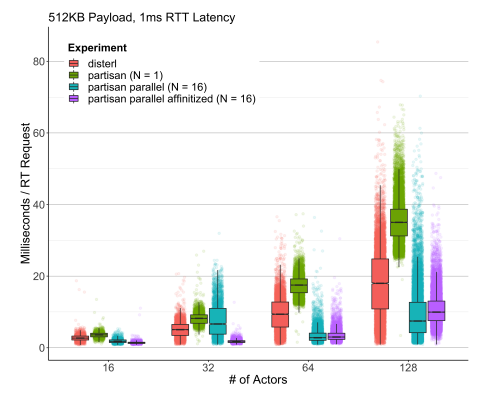


图1：分布式Erlang与Partisan的性能对比

如图1所示，我们首先显示了Distributed Erlang的基线配置。我们的结果表明，无论并发性如何，利用附加连接和关联通信都可以提高性能。拥有128个actor，512KB有效负载和1ms RTT，具有固定并行性的PARTISAN的性能比Distributed Erlang好1.69倍。考虑没有固定调度的PARTISAN，性能可提高1.90倍。在统一的工作量且没有网络瓶颈的情况下，固定调度比分布式Erlang产生了性能优势，但与纯随机调度相比，性能稍有下降。

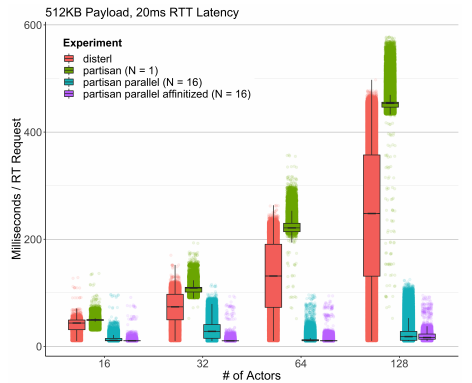


图2：高延迟下分布式Erlang与Partisan的性能对比

如果我们的应用程序分布在两个AWS可用区之间，而RTT却接近20ms，该怎么办？图2显示了这次运行我们较早的实验的效果，该实验位于不同节点上的参与者之间的RTT延迟为20ms。正如我们所看到的，随着等待时间的增加，系统可以利用更多的通信通道来并行化网络上的参与者之间的通信。具有128个actor，512KB有效负载和20ms RTT，具有并行性的PARTISAN的性能比Distributed Erlang好10.92倍。通过实现并行性，性能可比分布式Erlang提高到13.50倍。

针对第二个问题，为了了解特定网络覆盖对可伸缩性的影响，本文集中讨论可以将应用程序缩放到每个覆盖下使用Lasp实现的广告计数器应用程序的节点数。该应用程序是一个模拟的广告计数器，以针对《愤怒的小鸟》的Rovio计数器场景为模型。在此应用程序中，每个客户端都保留一个分布式计数器的副本，该副本在每次向用户显示广告时都会递增，并且其状态会定期传播到系统中的其他对等方。当达到一定的印象数时，广告被禁用并且不再显示给用户。所使用的分布式计数器是一种特殊的CRDT：仅增长计数器（G-Counter）。 G-Counter将每个客户端上的节点标识符映射到单调递增的计数器中。客户端增加其在地图中的位置，并且当合并状态从系统中的其他节点传播时，将为地图中的每个组件采用成对最大值。为了确定何时可以禁用广告，将根据地图中各个组成部分的总和检查下限：这表示广告已显示的总次数的下限。为了进行此评估，在相同区域和可用性区域中总共有70个m3.2xlarge Amazon EC2实例。 Mesos [16]被用来将这些机器中的每一个细分为更小的，完全隔离的机器。Mesos中的每个容器代表一个单独的Lasp节点，该节点使用PARTISAN与群集中的其他节点进行通信。每个计数器的增量间隔固定为10s，计数器的传播间隔固定为5s。配置了总展示次数，以确保实验在所有配置下都能运行30分钟。评估在客户端-服务器和对等覆盖上针对不同的群集大小进行，范围从32个一直到1,024个节点群集。

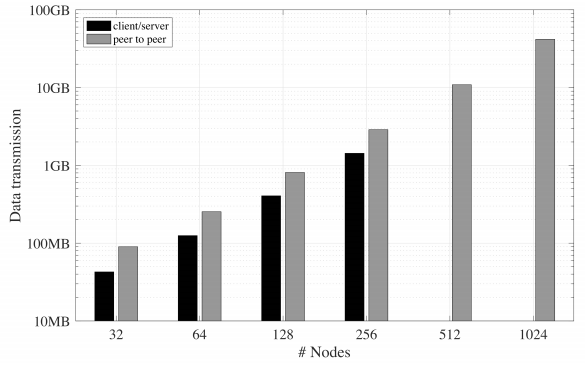


图3：比较不同群集大小（32到1024个节点）在客户端服务器和对等覆盖上部署的Lasp的数据传输量

图3展示了当我们将群集的大小从32个节点扩展到1024个节点时，完成实验所需的全部数据传输。对于较小的节点群集，就完成实验而言必须传输的数据量而言，客户端-服务器是更有效的网络覆盖。但是，这种提高的效率需要付出一定的代价：客户端-服务器配置无法扩展到超过256个节点。更具体地说，由于服务器崩溃失败，实验无法完成。由于无限制的消息队列而导致崩溃失败：当服务器无法足够快地处理来自客户端的传入消息时，Erlang VM会分配所有可用内存来存储消息队列。一旦实例可用内存不足，这种无限制的分配将导致Linux OOM杀手终止Erlang。面对节点故障，点对点更具弹性，从而使其能够支持更大的节点群集-多达1024个！但是，由于该原因，对等效率较低——覆盖所使用的通信链路的冗余导致其传输更多数据以完成实验。

四 结论

我们介绍了PARTISAN，这是一种替代的运行时系统，用于提高Actor应用程序的可伸缩性和减少延迟。 PARTISAN通过允许应用程序开发人员指定运行时使用的网络覆盖而无需更改应用程序语义，从而提供了更高的可伸缩性，从而将网络通信模式专用于应用程序。 PARTISAN通过结合三种主要的自动优化功能来减少消息等待时间：并行性，命名通道和固定调度。我们在Erlang中实现了PARTISAN，并显示PARTISAN通过运行时覆盖选择将系统可扩展的节点数量提高了一个数量级，吞吐量提高了38.07倍，延迟降低了13.5倍。在分布式Erlang上。