基于事件图模型的网格调度模拟

Grid Scheduling Simulation Based on Event Graph Model

张 千 梁 鸿 （中国石油大学 计算机与通信工程学院 山东东营 257061）

摘要：由于网格环境的复杂、动态和自治性等特点，研究网格任务调度时，高性能的网格模拟器是不可或缺的。该文引入了一个基于事件图模型的高性能模拟器 HyperSim，介绍了 HyperSim 的特点，通过对比其他模拟器说明使用 HyperSim 的理由。为了优化模拟速度，提出了网格任务调度的事件图模型，并给出了在 HyperSim 上的实现过程. 最后，通过实验证明了 HyperSim 在运行速度和性能方面的优势，并用其模拟了两种经典调度算法的实现，根据模拟结果对比了算法的性能。

关键词：网格计算 模拟器 HyperSim 事件图模型 分级式网格 调度算法

1 引言

网格技术是当今计算机领域的研究热点之一。在网格技术飞速发展的同时，网格计算中的任务调度问题也变得越来越重要。任务调度的目的是在网格环境中，同时考虑网格节点的计算性能、节点之间的通讯性能等参数，把不同的任务以最合理的方式分配给相应的网格节点去做。目前，围绕着网格计算中的任务调度算法，国内外已经做了大量的研究工作，先后提出了各种静态和动态的调度算法。验证这些算法面临的最大挑战就是网格环境的动态性，没有人可以实时的控制网格中的资源和运行的任务。其次，现有的网格实验环境规模通常都比较小，即使许多小规模的实验床通过中间组织连接起来，仍然达不到反映真实网格行为的规模。在这种情况下，网格模拟器成为检验网格调度算法可行性的最有效的工具。它不仅允许研究人员进行大量的实验，而且能够给出准确、稳定、实时的实验结果。

2 网格模拟器

网格模拟器的作用是模拟一个网格环境，在这个模拟的环境中研究不同的问题，比如可行性和调度问题。模拟器可以真实的模拟出真实环境中的各种应用场景，所以模拟结果具有真实性；通过分析在模拟器上的实验结果，网格研究者可以不断地改进设计。目前，在网格模拟领域，最具有代表性的模拟器（仿真器）有 GridSim、SimGrid 和 MicroGrid。

GridSim 是在 SimJava 的基础上开发的，可以进行离散的网格任务调度的模拟。该模拟器是用 Java 编写的，所以具有良好的跨平台性和可扩展性。但由于 Java 线程管理需要较大的系统开销，致使模拟的效率降低。SimGrid 是基于 C 语言开发的网格模拟器，可以精确建立网络环境的抽象模型。同基于 Java 开发的模拟器相比，它的运行速度要快的多。SimGrid 使用用户级线程来模拟资源，这种方法会受到线程交换和系统负载的限制。MicroGrid 允许应用程序在其虚拟网格环境中运行，通过截获应用程序中真实的资源调用函数，在虚拟资源上仿真其性能特性。它使用真实的网格应用程序，这样虽然可以使模拟结果更接近真实情况，但也需要更多的时间来完成一个应用程序，所以模拟的效率不高。

为了更广泛深入的研究网格环境，尽量缩短模拟时间，对高效的网格模拟器的需求就变得越来越迫切。因此，本文提出了将 HyperSim 高速模拟器应用于网格领域。HyperSim 相对其他模拟器而言最大的优点就是模拟速度非常快，可以用来模拟大规模的网格环境。

3 HYPERSIM 模拟器

HyperSim 模拟器主要应用于并行计算及电力调度领域。除了高速度之外，它还具有通用性，可扩展性和易配置等特点。HyperSim 是一个基于 C++ 开发的通

用离散事件模拟库，它提供了丰富的类，诸如事件发生器、统计分析器、自动轨迹仿真以及事件操纵器等来构造仿真环境。

HyperSim 遵循事件图模型，这种模型可以优化模拟速度、提高可扩展性。要建立某一系统的仿真模型，首先要设计出该系统的事件图模型，事件图至少应该包含一个节点事件，如果有两个节点事件，那么它们之间通过带箭头的直线表示关联。图 1 表示了最基本的事件图模型，它描述了两个事件以及事件 A 到事件 B 的状态转换，图中，当前事件是 A，如果条件 i 成立，事件 B（预期事件）就会在 t 个时间段后发生。预期事件的属性集由事件发生器自动产生。

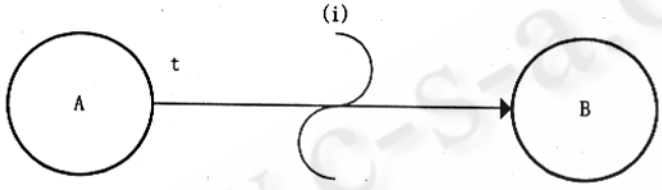

图 1　基本的事件图模型

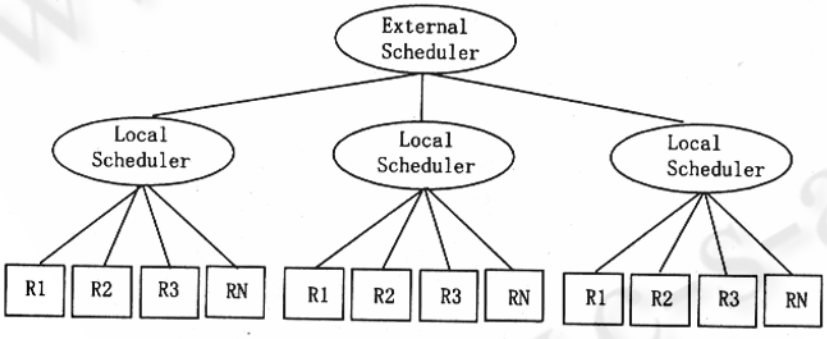

图 2　分级式网格结构

模拟开始时，通过事件队列初始化模拟器，事件队列中至少要包含一个预期事件。事件队列中的事件会在模拟过程中按照发生顺序自动离队。当事件队列为空时，整个模拟过程就结束了。事件队列中的所有的事件在执行过程中，都会写入到日志中。

4　网格调度建模

为了更方便的研究网格调度，可以将网格系统按照调度中心的位置不同划分为两种类型：集中式网格和分级式网格。集中式网格中的调度器可以直接访问所有资源。但是在分级式网格系统（如图 2 所示）中，调度器只能与本地资源管理器进行交互，不能直接控制本地资源。分级式网格应用实例有很多，例如许多个机群通过网格节点相连后形成的结构就是分级式网格。一般来说分级式结构更好一些，因为它可以使加入到网格中的组织更好的管理本地资源。

无论是集中式网格还是分级式网格，应用事件图模型都可以有效的为其建模。图 3 给出了分级式网格结构的事件图模型。该模型包括两种调度器：ES（External Scheduler，全局调度器）用作网格层的调度器，LS（Local Scheduler，本地调度器）用作机群层的调度器。

如图 3 所示，起始事件是 INPUT，它的作用是初始化系统。INPUT 事件之后产生了 ENTER 事件和 SCHEDULE 事件。ENTER 事件代表着将网格任务提交到调度队列中。现有的网格任务调度器可以分为在线模式（on－line）和批模式（batch－mode）两种。对于在线调度器，任务一到来就加以映射，SCHEDULE 事件将立刻被调度。而批调度器则是把任务收集起来等映射事件到来后，才对这些任务进行集中映射。所以 ENTER 事件在经过随机的任务到达时间间隔（由 Ti 标识）后才能执行，相应地 SCHEDULE 事件每隔 Ts 就要被重新调度一次，同时所有新到达的任务也被执行调度，这个调度过程通过调用 START 事件来完成. 任务执行完毕后，启动 FINISH 事件。

EENTER，ESCHEDULE，ESTART，和 EFINISH 事件都是网格层的事件。ESTART 事件需要把任务提交到机群的调度器而不是执行调度任务。图中的 STAGE_IN，STATE_EXE，EXECUTE 和 STAGE_OUT 事件代表着在跨越广域网连接网格层和机群层调度器的情况下，任务的分级输入、执行以及分级输出。

事件图模型建立之后，根据模型中的事件流程，依次建立模拟器中的实体。首先通过 HyperSim 工具包实现 Host 类，Cluster 类和 Grid 类，它们分别代表不同类型的网格资源。Host 代表单独的或者是有多个处理器的系统，Cluster 代表通过高速网络连接起来的 Host，Grid 则表示由广域网连接起来的 Cluster。

任务的运行时间是由任务数量、运行速度和当前

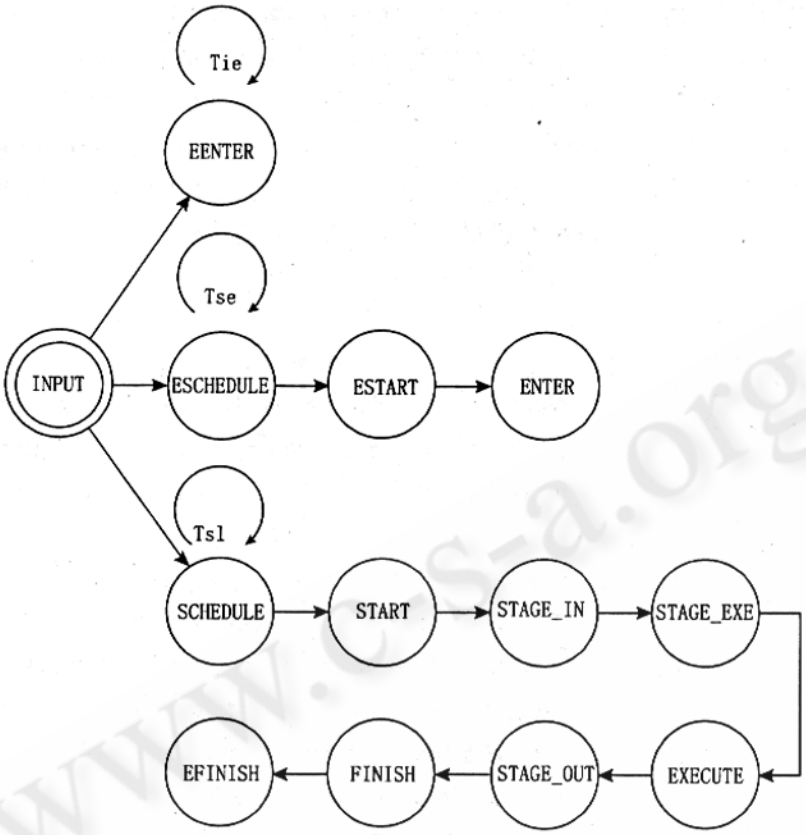

图 3　分级式网格的事件图模型

已用资源的平均负载这三个因素决定的。令 w 表示任务量，ε 表示任务运行速度，λ 表示当前系统的平均负载，那么任务运行时间 Te 的表达式如等式 1 所示。

$$T_e = \frac{W}{\varepsilon}(1+\lambda) \quad (1)$$

工作负载可以从指定文件中读取，也可以根据实际的负载的迹（workload trace）得出。另外，Cluster 配置可以自动产生或者以文件的形式在模拟器中给出。自动配置需要用户指定一些参数，如 Host 的数量，运行速度，以及平均负载等。对于 Grid 配置来说，必须明确指定每个 Cluster 的配置情况和网络状况。调度算法相对于模拟器来说是独立的。每一种调度算法都是基于不同的动态链接库执行的。这意味着用户可以方便的实施多种调度算法，而不必对模拟器作任何修改。

5　试验及讨论

为了更好地说明 HyperSim 在速度和性能方面的优势，我们使用不同的模拟器对网格调度算进行模拟。所有的测试均在一台装有 Linux 系统的 PC 机上进行，使用 1G 的 CPU 和内存，并且主机都处于空载状态。我们选用 GridSim3.2 和 SimGrid2.93 来进行比较试验。为了评估模拟器的性能，使用 MET（Minimum Execution Time，最短运行时间）调度算法分别在 GridSim，SimGrid 和 HyperSim 三种模拟器上实施，通过模拟时间来对它们的性能进行比较。MET 是一种在线调度算法，无论任务何时到达，都可以即时进行调度。MET 算法首先对所有机器运行某一任务的时间进行估测，然后将该任务调度到使之最快完成的主机上执行。

图 4 给出了任务数固定为 16384，逐步增加网格资源数量时三种模拟器的运行时间。网格资源数将按如下规律增长：1，2，4，8 直至 16384。从实验结果中可以发现，GridSim 所模拟的资源数量不能超过 512 个，这是因为使用的资源量过大时会产生线程创建的错误。如图 4 所示，虽然 SimGrid 和 HyperSim 都能模拟资源不断增加的情况，但是 HyperSim 的运行速度明显的比 SimGrid 要快出 10 倍左右。

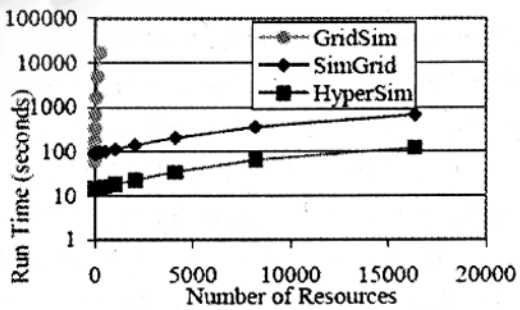

图 4　基于固定任务数的比较结果

然后，将资源数固定在 256，逐渐增加网格任务的数量，通过这个实验再来评估模拟器的性能。之所以将资源数控制在 256 是因为 GridSim 只能在 512 个资

源以内进行模拟。模拟结果如图 5 所示。在速度方面，仍然是 HyperSim 遥遥领先，它的运行速度相当于 GridSim 的 1000 倍，SimGrid 的 10 倍。很明显，Java 线程管理器导致了 GridSim 的系统开销非常大。而且，GridSim 在模拟过程中始终分配固定数目的资源，即使任务数量减少也是如此，导致内存的使用效率降低。

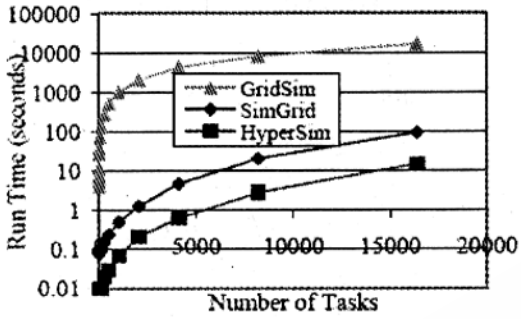

图 5 基于固定资源数的比较结果

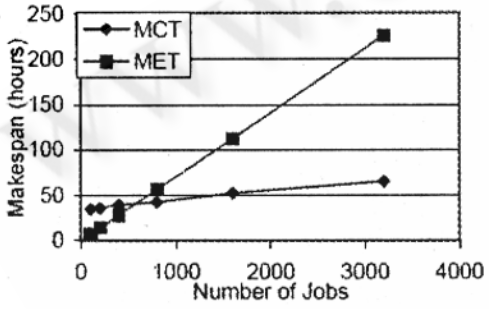

图 6 MET 和 MCT 的 Makespan（总执行时间）

虽然 SimGrid 的模拟结果与 HyperSim 很相似，但是两者有本质的区别。SimGrid 只允许用户模拟事先定义好的中止条件，例如当一个或者是所有任务运行结束时。这样虽然简化了任务编程的复杂度，但需要频繁地进行条件检测，使得代码更为复杂。实际上降低了 SimGrid 模拟器的运行速度。与此相反，HyperSim 在建立事件模型时比较复杂，而编码则相对简单，并且代码的执行效率较高。

网格环境下，高效的调度策略或算法可以充分利用网格系统的处理能力，从而提高应用程序的性能。以完成时间（complete time）为优化目标的任务－资源映射式 NP 完全问题，所以需要辅助的启发算法。

6 结论

本文引入了 HyperSim 高速模拟器，建立了网格调度的事件图模型，并阐述了在 HyperSim 中如何利用这种模型构造仿真环境。通过与其他两个网格模拟器 GridSim 和 SimGrid 进行对比，证明了 HyperSim 具有速度快、效率高、适合多资源模拟的优点。最后，应用 HyperSim 模拟了两种启发式调度算法的实现，通过模拟结果说明了两种算法的优缺点。

参考文献

1 Aida, K., A. Takefusa, H. Nakada, S. Matsuoka, S. Sekiguchi and U. Nagashima, 2000. Performance Evaluation Model for Scheduling in a Global Computing System[C], The International Journal of High Performance Computing Applications, 14(3).

2 T. Braun, H. Sigel, N. Beck, L. Boloni, M. Mashesweran, A. Reuther, J. Robertson, M. Theys, B. Yao, D. Hensgen. A comparison study of static mapping heuristics for a class of meta－tasks on heterogeneous computing systems[C], In 8th IEEE heterogeneous computing Workshop (HCW '99), pages 15－29, Apr. 1999.

3 BHARADWAJ V, CHOSE D, ROBERTAZZI TG. Divisible Load Theory: A New Paradigm for Load Scheduling in Distributed System[J], Cluster Comput., 2003, 6(1):7－17.

4 A Abraham, R Buua, B Nath. Nature Heuristics for Scheduling Job on Computational Grids[C], In: ADCOM 2000, Cochin INDIA, 2000－12:45－52.

5 L. Wang, H. J. Siegel, V. P. Roychowdhury, and A. A. Maciejewski. Task matching and scheduling in heterogeneous computing environments using a genetic－algorithm－based approach[J], Journal of Parallel and Distributed Computing, 1997, 47(1): 1－15.

6 R. F. Freund and H. J. Siegel. Heterogeneous processing[J], IEEE Computer, 1993, 26(6):13－17.

7 O. Ibarra and C. Kin. Heuristic algorithms for scheduling independent tasks on nonidentical processors [J], Journal of the ACM, 1997, 77(2): 280－289.