第 34 卷 第 2 期 计 算 机 工 程 2008 年 1 月
Vol.34 No.2 Computer Engineering January 2008
·软件技术与数据库· 文章编号：1000—3428(2008)02—0086—03 文献标识码：A 中图分类号：TP309

基于生物计算的分布式计算系统

张祖平，王 丽
（中南大学信息科学与工程学院，长沙 410083）

摘 要：分布式计算技术提供了充分利用现有网络资源的有效途径。该文论述了基于解决生物计算中难解问题的具有开放接口的分布式并行计算系统的设计与实现技术。系统兼有开放式、异构性、容错性与易用性等特点。讨论了系统的容错性机制、检查点策略及任务调度算法。对 Motif Finding 问题的求解验证表明，分布式并行计算机制能大大缩短问题的求解时间，为计算领域的难解问题提供有效的解决途径。
关键词：分布式计算系统；容错性；任务调度

Distributed Computing System Based on Biology Computation

ZHANG Zu-ping, WANG Li
(School of Information Science & Engineering, Central South University, Changsha 410083)

【Abstract】 Distributed computing technology provides an effective way to adequately utilize network resources. This paper presents a universal distributed parallel computing system, which is based on solving NP-hard problem in computational biology and provides an open interface for client to upload application and call service. The system is characterized with openness, isomerism, fault tolerance and usability. The paper focuses on the fault-tolerant mechanism, checkpoint strategy and task schedule algorithm. Experiment for Motif finding problem shows that the system can shorten running time sharply. And it's also an effective way to solve other hard problems in computing area.
【Key words】 distributed computing system; fault tolerance; task schedule

在生物学的研究中，大量的课题都是NP-难问题，这些问题的解决往往需要大规模的计算，而高性能超级计算机价格昂贵且具有局限性。随着硬件技术的发展，越来越多的个人计算机构成了庞大的网络，据统计，截至 2000 年，与Internet 连接的计算机超过 3 亿台，每台机器有 80%～90%的CPU资源闲置，怎样组织和利用这些闲置资源进行分布式计算，解决生物计算或其他领域的难解问题受到越来越多的关注[1]。本文结合Java的平台无关性与Web的易用性而实现的分布式计算系统，采用志愿者模型[2]，具有动态可用性与资源异构性等特点。此外，该系统还有以下特点：(1)开放性：经授权的用户可以从网络向系统提交计算任务，利用本系统的计算资源快速获得结果。(2)易用性：提供简单友好的界面，志愿者只需在第一次申请时安装JRE和客户端软件，其余工作由客户端软件自动完成。(3)易编程性：接口简单清晰，方便用户编程，无需关注调度、容错等细节。(4)异构性：能适应多种系统环境，并通过防火墙；(5)容错性：系统能够容忍和处理志愿机的不确定性等造成的错误。

1 系统设计

1.1 系统设计方案

系统采用 Java 语言编程实现(结构见图 1)，选用 Java Application 与 HTTP 协议，跨越多种平台并解决跨防火墙通信问题，通信数据用 XML 封装。系统主要构成如下：

(1)客户机端。客户机是计算资源的请求者，它向服务器提交问题，等待结果的返回。用户编写满足系统接口的应用程序，由 Web 页面上传，服务器将其自动部署为计算服务，然后客户机上传任务调用服务，服务器分解任务并分配到各个工作机上计算，计算完成后将结果整合返回给客户机。

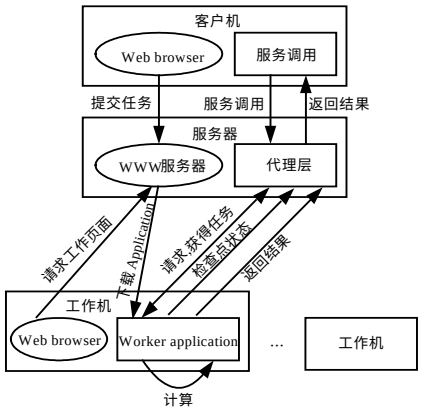

图 1 系统结构

(2)工作机端。工作机提供空闲计算资源，其工作流程是一个循环的过程：向服务器申请任务，开始计算，返回计算结果。工作机首次申请任务时下载应用程序到本地，随后程序自动向服务器申请任务，进行计算，返回结果，直到志愿者关闭计算机或接到任务完成通知才终止申请与计算。另外，由于系统采用了检查点机制，工作机在计算过程中需要暂停计算，保存中间状态，然后发送到服务器。

(3)服务器端。面向客户机，接收提交的任务并返回结果；

基金项目：国家自然科学基金资助项目(60433020)
作者简介：张祖平(1966－)，男，副教授、博士，主研方向：参数计算及应用，网络容错路由算法，大型数据库技术；王 丽，硕士研究生
收稿日期：2007-01-25 **E-mail：**zzpcsu@hotmail.com

面向工作机，接收请求，分配任务，接收返回的检查点中间状态和任务结果。服务器提供客户机和工作机访问的 Web 接口，另外，代理层的工作——将应用程序分解的任务集合存储；响应工作机的请求，采用合适的任务调度算法分配任务；接收工作机返回的结果，并对结果进行验证；接收保存工作机发送的检查点状态信息，在需要的时候实现计算的迁移也由服务器完成。

1.2 系统模块与流程

服务器和工作机两部分采用 Master-Worker 模型，系统模块见图 2。

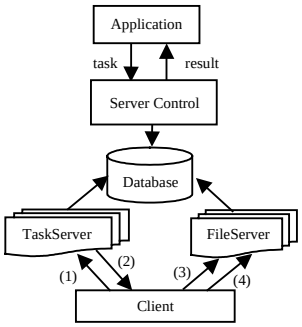

图 2 系统模块设计

Application 模块负责完成算法中的串行部分，即将问题分解成独立的并行任务(task)和整合返回结果(result)；ServerControl 是服务器端的主控模块，负责与 Application 模块的交互：提供接口给 Application 将其产生的任务插入数据库，并调用 Application 默认实现的接口将结果上报给 Application；TaskServer(任务服务器模块)和 FileServer(文件服务器模块)实际上是 Servlet 实现的 HTTP 服务器，前者负责接收工作机的任务请求，执行任务调度；后者负责程序和数据文件的下载以及结果文件的上传。Database 中的 Task 表记录了任务的详细信息，包括：任务 ID，任务状态，发送时间，时间限制，检查点信息等。服务器端的 3 个模块：ServerControl，TaskServer 和 FileServer 是通过访问、修改 Task 表来交互的。Client 是运行在工作机上的客户端软件。

图 2 中的标号(1)～标号(4)代表的工作机工作过程描述如下：

(1)Client 发送 XML 封装的请求消息 task_request.xml 到 TaskServer，请求消息中包含 Client 所在工作机的 IP，计算机的基本配置等。

(2)TaskServer 检查请求的合法性并判断志愿机是否符合运行任务的最低要求后，执行任务调度，将分配任务的信息，包括：任务 ID，要运行的程序 ID，任务需要的数据文件 ID 等，封装成响应消息 task_reply.xml 发送给 Client。

(3)Client 根据 task_reply.xml 的内容，向指定的 URL 发送 HTTP 的 GET 命令下载相应的程序和数据文件。

(4)计算结束后 Client 通过 HTTP 的 POST 命令上传结果文件到 FileServer。

2 系统关键技术及实现

分布式计算环境中，多台机器要协同工作，会涉及机器的组织、问题的分布等问题；网络环境中有传输延迟、容错等问题。下面就几个主要问题给出实现机制。

2.1 容错机制

由于网络本身的不可靠性和志愿机加入离开的任意性，系统必须考虑容错。

由于适合分布式并行计算的任务应该是粗粒度的，具有很高的计算通信比，因此采用超时任务重分配机制。对于已经分配的任务，在规定的时间内若没有结果返回，就认为分配了该任务的工作机已经退出系统或者不能完成任务，修改数据库中该任务的状态为未分配，等待下一次分配。每个任务的时间限制由 Application 指定，ServerControl 定时扫描 Task 表，根据任务的发送时间和时间限制找到超时任务进行处理。

但是此机制可能引发多个工作机上报同一个任务结果，因此，系统只接受最早返回的结果。工作机在申请新任务的 task_request.xml 中加入<result_report>来报告完成任务的情况：

```
<result_report>
    <task_id>123</task_id>
    <result_state>0</result_state>
    <!-- 0 represents success, -1 represents failure -->
    <total_cost_time>45.34</total_cost_time>
    <!-- if successfully processed record total cost time -->
</result_report>
```

TaskServer 解析<result_report>，如果任务成功完成且还未得到该任务的结果，通知工作机上传结果；否则通知不上传。通知消息是在 task_reply.xml 中的<result_ack>表达的：

```
<result_ack><task_id>123</task_id>
    <upload_result>0</upload_result>
    <!-0: should upload result，-1: no need -->
</result_ack>
```

工作机上的 Client 解析<result_ack>，需要上传则上传，否则删除结果文件。

另外，系统还考虑了 Byzantine faults，包括：无意的随机错误，例如数据丢失、失败的网络连接或者处理器错误和作弊，恶意的攻击等。对此，系统采用的机制如下：

(1)简单任务的正确性验证。某些应用的并行任务的结果很容易验证正确与否，如解某个复杂方程，只要将结果代入方程就可以验证；对于那些只有少量解的搜索问题，例如货郎担问题，可以给出判定函数来判断结果的正确性。

(2)Majority-voting 冗余技术验证。由于函数级验证的适用范围有限，因此系统将任务分配 $2m$-1 次，要求得到的结果中至少有 m 个是一致的。参数 m 由应用程序根据准确度要求设定。

2.2 检查点

由于并行任务是粗粒度的，需要长时间的计算，一旦出错将浪费此前的大量计算，为避免这种损失采用检查点(checkpoint)机制。文献[3]证明在故障条件下，如果不用检查点，程序平均执行时间随其有效执行时间呈指数增长，而使用固定间隔的检查点时，呈线性增长。检查点技术不仅能实现故障恢复，还能实现计算迁移：利用检查点保存计算在某台机器上的运行状态，然后在其他机器上恢复运行。

根据中间状态保存的地点，检查点机制分为两种：本地检查点模型和网络检查点模型[4-5]，前者将中间结果保存在本地，后者将中间结果发送回服务器，本文采用后者。

设置检查点首先要考虑设置时机。检查点设置是周期性的，由应用程序决定什么时候暂停计算及保存哪些状态信息。因此，应用程序要在适合进行检查点记录的地方插入语句判断距最近一次设置检查点是否有一个周期的时间，若是，则记录状态信息；否则继续计算。判断是否到检查点时间的方法由系统客户端提供。

为实现计算重启及计算迁移，应用程序要实现重启计算

方法，中间状态信息要随着任务对象一起发送到工作机端。

检查点机制要求应用程序实现检查点的接口：

```
public class BasicTask    //基本任务类
private object state_info; //记录计算最新的中间状态。对于一个
//刚获得的任务，该值由任务调度模块写入。若其值为 null，则计算
//重头开始，否则计算从断点开始
public long start_time;   //计算开始的时间
private CheckComThread check_com;   //负责进行检查点通信的
//线程
boolean time_to_checkpoint() //判断距上一次检查点是否已经有
//一个周期的时间间隔
void do_checkpoint(Object state_tmp) //在记录检查点之后调用，
//首先将新的中间状态信息 state_tmp 与 state_info 比较，若不同则启
//动通信线程 check_com 发送给服务器并替换 state_info
abstract BasicResult resume() //由应用程序员实现，完成从任务
//的 state_info 中读取状态信息继续计算
```

2.3 任务的调度

采用 advanced eager-scheduling 调度算法，即考虑了容错(指 majority-voting)与检查点机制的 eager-scheduling 算法。

eager-scheduling 算法描述：工作机的每次申请只能获得一个任务，运算能力强的工作机由于完成任务快将会获得较多的任务。当要分配的任务少于工作机数量时，将已经分配的未完成的任务重新分配出去，并处理机器失败的情况。

advanced eager-scheduling 算法描述：

进行m-voting，每个任务至少要分配 $2m$-1 次，对得到的 $2m$-1 个结果验证。实际上，更为有效的方式是m-first voting[5]——不等到有 $2m$-1 个结果才开始判断，而是从有m个结果返回后开始比较，只要有m个结果一致，就认为该任务得到了正确的结果。针对此算法，每个任务首先分配m次，如果这m个结果不一致，再分配第 $m+1$次，直到有m个结果一致为止。

为记录任务的状态、分配情况等，设计类 TaskUnit 封装任务信息，ComputeUnit 封装每一次计算(一个任务分配一次即是一次计算)的信息。每一个 TaskUnit 对象都有一个 ComputeUnit 队列，将一个任务分配一次，就在它的 ComputeUnit 队列中新增一个 ComputeUnit 对象。TaskUnit 定义了多种状态，如 UNSTARTED(任务还未开始处理)、IN_PROCESS(任务等待继续分配)、WAIT_RESULTS(等待某些计算结果返回)、WAIT_VALIDATED(等待结果的验证)、HAS_DONE(任务已经完成：得到验证后的结果或者计算失败)。

在进行任务调度时，根据 m-first voting 的思想，IN_PROCESS 状态的任务计算的次数少于 m，由于希望尽快得到一个任务的结果，而不是让更多的任务开始计算，这样可以尽早从任务池中删除完成的任务，因此首先要分配的是这类任务，然后是处于 UNSTARTED 的任务。调度算法还要处理计算失效(结果超时未返回)的任务，即将 WAIT_RESULTS 状态超时的任务重新分配，重分配不是对任务重新分配计算(新创建 ComputeUnit 对象)，而是对未完成的计算的继续——将计算最新的状态信息和任务一起发送给工作机，工作机从断点继续计算。

3 结束语

通过测试在 k=10(序列个数)，n=82(序列长度)的范例序列中寻找(15, 4)-Motif(Motif 长度为 15，允许 4 个操作)，运行结果如表 1 及图 3 所示。

表 1 多台工作机运行时间测试

客户机数量/台	总运行时间/h
1	0.74
2	0.46
3	0.25
4	0.24
5	0.24
6	0.17
7	0.13

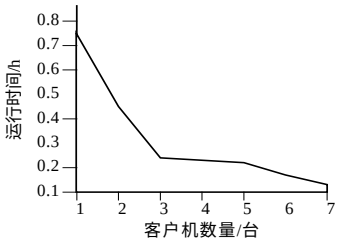

图 3 多台工作机运行时间测试曲线

从测试结果来看分布式计算机制对程序运行时间有极大的改进。系统不仅为生物计算中的难解问题提供了解决的方法，也为计算领域中可以将计算任务分解的难解问题提供了有效的解决途径。

参考文献

[1] 王建新, 黄 敏, 李绍华. 基于任务树的分布式计算平台的设计与实现[J]. 小型微型计算机系统, 2006, 27(5): 940-944.

[2] 梅 皓, 沈志宇, 廖湘科. 基于 Java 的分布式并行计算关键技术[J]. 计算机工程与科学, 2000, 22(2): 103-106.

[3] Duda A. The Effect of Checkpointing on Program Execution Time[J]. Information Processing letters, 1983, 16(4): 221-229.

[4] Germain C, Fedak G, Neri V, et al. Global Computing Systems[C]// Proc. of the 1st IEEE/ACM International Symposium on Cluster Computing & the Grid, [S. l.]: ACM Press, 2001: 582-587.

[5] Sarmenta L F G, Volunteer Computing[D]. Massachusetts: Dept. of Electrical Engineering and Computer Science of MIT, 2001.

(上接第 77 页)

参考文献

[1] Witten I, Moffat A. Managing Gigabytes[M]. San Francisco, CA: Morgan Kaufumann Publishers, 1999.

[2] Lester N, Zobel J, Hugh E. Williams. In-place Versus Rebuild Versus Remerge: Index Maintenance Strategies for Test Retrieval System[C]//Proc. of the 27th Conference on Australasian Computer Science. Dunedin, New Zealand: [s. n.], 2004, 26:15-23

[3] Büttcher S, Charles L. Hybrid Index Maintenance for Growing Text Collections[EB/OL]. (2006-01-02) http://stefan.buettcher.org/papers/buettcher_2006_hybrid_index_maintenance_2.pdf.

[4] Lester N, Moffat A, Zobel J. Fast On-line Index Construction by Geometric Partitioning[C]//Proc. of the 14th ACM Conference on Information and Knowledge Management. Bremen, Germany: [s. n.], 2005.

[5] Zoble J. Inverted Files Versus Signature Files for Text Indexing[J]. ACM Transactions on Database Systems, 1998, 23(4): 453-490.

[6] 关 毅, 王晓龙. 现代汉语计算语言模型中语言单位的频度-频级关系[J]. 中文信息学报, 1999, 13(2): 8-15.