第43卷 第10期 西 安 交 通 大 学 学 报 Vol. 43 №10
2009年10月 JOURNAL OF XI′AN JIAOTONG UNIVERSITY Oct. 2009

一种应用于远程并行程序调试系统的新型消息聚集机制

赵海祥，伍卫国，赵增，李海龙
（西安交通大学电子与信息工程学院，710049，西安）

摘要：针对并行程序调试相对复杂的问题，提出了一种新的消息聚集机制，实现了适用于调试大规模并行程序的远程源码级调试原型系统.核心的消息聚集策略包括：消息收集和传送的树形控制结构；合并调试命令的返回结果消息.通过改进固定分支数的树形结构，将树分成2层，节点内部构成一棵树，节点之间构成一棵树，以此最大程度地减少节点间的消息传递量.系统底层集成了串行调试工具GNU gdb，以完成单个进程的调试.另外，设计了一种能同时满足4类gdb结果情况的消息合并方法，在最终的结果消息中保留了gdb的原始输出数据.实验结果表明，调试原型系统能满足用户的基本要求，并简化了远程计算资源的方式，屏蔽了复杂的服务器硬件结构及处理流程.

关键词：远程计算；消息聚集；并行程序调试；树形结构
中图分类号：TP309 **文献标志码**：A **文章编号**：0253-987X(2009)10-0027-05

A New Mechanism of Message Aggregation Applied to Remote Parallel Program Debugging System

ZHAO Haixiang, WU Weiguo, ZHAO Zeng, LI Hailong
(School of Electronics and Information Engineering, Xi′an Jiaotong University, Xi′an 710049, China)

Abstract: A new mechanism of the message aggregation is proposed for debugging large-scale parallel programs. A remote source level debugging prototype system that is suitable for debugging large-scale parallel programs is created. The core of the message aggregation strategy includes: a tree structure that is used to aggregate and transfer the messages, and the resulting message returned by debug combination commands. Through the improved tree structure with fixed number the tree can be partitioned into two layers: a tree formed by nodes, and a tree formed by controllers within nodes, so that the amount of message passing between nodes can be greatly reduced. The serial debug tool GNU gdb is integrated into the bottom of the system to carry out the single process debug. A method of message combination is designed for satisfying the four kinds of gdb results. Meanwhile, the original output data of gdb are retained in the resulting information. Experimental results show that the debugging prototype system satisfies user's basic demands, simplifies the remote calculation, and shields the complicated hardware structure and processing flow.

Keywords: remote computing; message aggregation; parallel program debugging; tree structure

并行程序调试相对于串行程序调试更为复杂，缺乏有效的调试工具是阻碍并行程序设计和并行计

收稿日期：2009-03-10. 作者简介：赵海祥(1983—)，男，博士生；伍卫国(联系人)，男，教授，博士生导师. 基金项目：国家自然科学基金资助项目(60773118)；国家高技术研究发展计划资助项目(2006AA01A109).

算技术发展的重要障碍之一[1]. 随着网格计算等分布式处理技术的发展,人们对跨网络的并行程序调试工具的需求显得更为迫切. 并行程序的调试是一种交互性很强的过程,数据传输密集,处理机间传递的调试信息量会随着进程数的增加而增大. 当进程数扩展到成百上千时,需要传递的消息量非常大[2],尤其在远程环境下,会严重影响系统的响应速度. 采用何种策略能更快地聚集数据,减小网络传输的数据量,缩短调试动作的响应时间,使系统能快速地反馈调试结果,是当前迫切需要解决的问题.

本文以此为背景,提出了一种高效的消息聚集机制,并设计了一种能够支持远程调试并行应用程序的调试器框架.

1 调试器的系统结构

调试器的系统结构如图 1 所示,其整体上分为调试客户端和调试服务器端 2 部分,它们之间通过 Internet/Intranet 互连. 客户端提供了一个图形操作界面,接受用户的调试操作,将相应操作转换成调试命令并经过编码后发送到服务器端. 服务器端处理调试命令,将调试结果返回给客户端. 客户端在接收到调试结果以后,异步地更新用户界面的状态.

客户端构建于 Eclipse 平台,以插件形式发布,并采用模式-视图-控制器(MVC)模式设计,主要模块包括:①ActionHandler;②EventHandler;③Debug Client;④Communicator;⑤其他模块. 模块①分析用户操作类型,调用相应的处理模块. 模块②是当后台有新的数据到达或后台状态发生改变时,以事件的形式通知用户,即从事件队列里取出未处理的新事件,分析事件类型,取得事件参数并反映到用户界面(UI)上,同时刷新 UI 的状态和数据. 模块③是客户端核心模块,有 2 大功能:将 UI 发出的调试操作经编码后插入命令队列,并交由通信模块发送;将远程服务器的结果数据经解码分析后封装成事件,按照不同类型插入不同的事件队列,激活事件,并交由该事件的相应监听者处理,最后将处理结果反映到 UI 之上. 模块④可与服务器端通信,处理底层 I/O 操作,对命令进行打包,调用 Socket 通信的应用程序接口(API)发送命令和参数,接收结果数据,并交由上层模块进行处理. 模块⑤进行用户身份认证,向用户提供窗口、工具条、按钮等操作,显示监视操作结果,进行工程项目管理、并行环境的配置和管理,记录调试器中各组件的执行情况.

服务器端功能模块与客户端相对应,其与客户端各模块配合共同完成调试任务. 集中式控制总体上分为 2 层:全局管理者和局域管理者. 前者是服务器端的管理者,运行于主节点(集群系统的控制节点)之上,既负责与客户端的通信,又负责控制各个局域管理者的启动、运行,接收来自客户端的命令并分发给各局域管理者,收集来自局域管理者的数据并返回给客户端. 后者运行于各计算节点之上,是各个本地调试进程的管理者,负责控制本地调试器的启动、运行,收集本地调试器的输出数据等. 各节点上的局域管理者之间通过底层并行编程环境(如 OpenMPI[3])互连,构成并行计算环境.

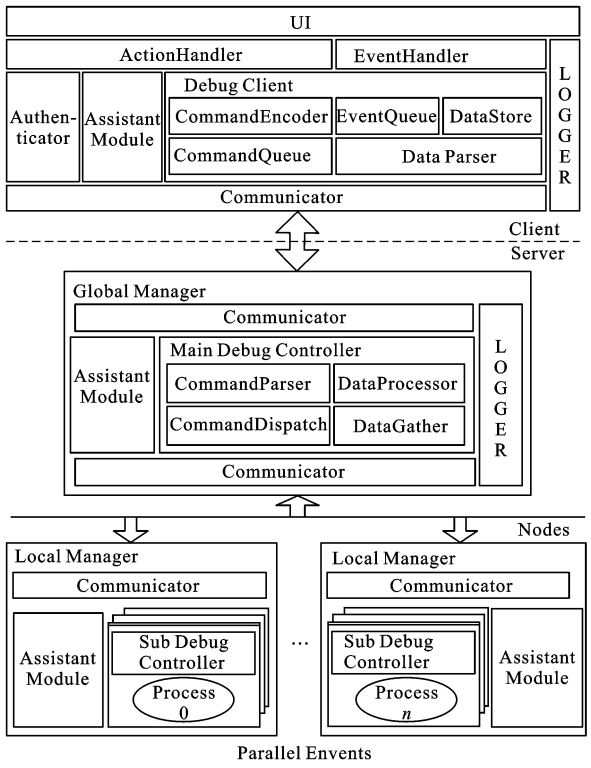

图 1 系统功能模块划分

2 树形控制结构

服务器端调试模块的主体是调试控制器,包括主控制器和子控制器. 主控制器是客户端与子控制器之间连接的桥梁,子控制器控制着本地 gdb 进程的运行. 为了提高系统在大规模程序调试时的运行效率,需将所有的调试控制器从逻辑上组织成一个树形结构,包括 2 个层次的树,即每个节点内部的控制器构成一棵树,这些树的根合在一起再加上主控制又构成另外一棵树,如图 2 所示. 这样做是为了将一个节点上所有进程的调试数据经全部收集、合并之后再向上层发送,以此减少节点之间通信的次数

和通信量. 根据树形的位置，子控制器又可以分为 2 种：中间控制器和叶子控制器. 中间控制器位于主控制器和叶子控制器之间，除了要控制本地的串行调试器之外，还要负责上下层控制器之间的通信，接收上层控制器传来的调试命令并发送给下层，收集下层控制器的调试结果并返回给上层. 叶子控制器位于树的底层，只负责本地串行调试器的控制以及与上层之间的通信，各叶子独立工作.

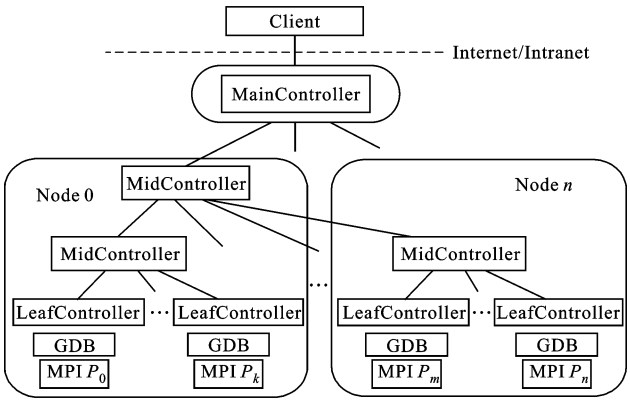

图 2　服务器端调试控制架构

树中数据的传递流程如下：主控制器接收客户端的调试命令并发送给它的直接子控制器；子控制器向其下层子控制器发送调试命令，同时将命令传递给本地被控制的 gdb 调试进程. 命令沿着树枝向下传送，直到叶子子控制器把命令传给 gdb，并获得 gdb 的执行结果. 结果数据收集过程与调试命令传递过程相反，即：各叶子把结果数据发送给父控制器；父控制器对这些数据进行合并操作，并将合并结果发送给上一层父控制器. 上层父控制器进行数据合并，并将合并结果发送到其上一层，直到到达主控制器. 主控制器在收集到所有结果数据后，做最后一次合并，打包相应的参数并发送给客户端. 这样，一个调试命令的响应过程就完成了. 需要注意的是，为了实现进程集调试，调试命令包含了进程集标识，表明本次命令需要执行哪些 gdb 进程. 各级控制器在收到命令之后确认自己在进程集中的位置，并将命令传递给本地的 gdb 进程，否则将命令传递给子控制器.

树形结构由全局管理者建立、维护，主控制器位于直接与客户端通信的节点之上，子控制器的数量和所处位置取决于当前调试会话中的调试进程数、节点数和树的分支数. 与树形结构对应的传统结构是由一个主控制器直接控制全部的子控制器[1,4-5]，这种结构设计相对容易，但消息传递的复杂度为 $O(n)$，其中 n 为进程数. 树形结构可将复杂度降至 $O(\log_b n)$，其中 b 为树的分支数，采用这种结构可以减少根节点的负载. 从理论上看，当调试大规模并行程序时，树形结构可以更充分地利用系统的并行性，提高聚集效率. 文献[6]的树形消息聚集结构中，树的分支数固定，所有的处理器同等对待，中间控制器的位置仅由进程数和树的分支数决定，这样就有可能在一个控制器的子控制器中既有本地的又有远程的. 这种控制器与其 2 种子控制器之间的通信速度并不相同，如果同等对待势必会影响系统的整体性能. 本文改进了这一点，以节点为单位进行聚集，由此降低了节点间的通信次数和通信量，但是本设计会增加树的高度. 假如节点数为 m，并行进程数为 n，树的分支数为 b，则一个节点的进程数为 $\lceil n/m \rceil$，节点内部树的高度为 $\lfloor \log_b \lceil n/m \rceil \rfloor + 1$，节点间树的高度为 $\lfloor \log_b m \rfloor + 1$，整棵树的高度为 $\lfloor \log_b \lceil n/m \rceil \rfloor + \lfloor \log_b m \rfloor + 1$，那么用固定分支数的方式设计的树高为 $\lfloor \log_b n \rfloor + 1$，其高度至少比本文减小了一个数量级.

此外，短消息聚集技术也是消息聚集中的关键技术. Pham 等人[7]研究了短消息聚集在低延迟消息传递环境下的性能，他们指出，消息聚集的依据是：如果发送一个大小为 M 的消息，会比发送 m 个大小为 n（$mn=M$）的消息所花费的时间更少. 对细粒度应用来说，短消息聚集的作用能明显减小网络通信量，但是短消息聚集带来的负面作用是接收端长时间处于等待状态，这是一个相互抵消的过程，所以聚集时间不能太长. 在高性能集群环境下进行并行模拟时，聚集长度最好是 156 B. Pham 等人还指出，优化聚集策略需要仔细考虑底层的硬件结构和通信子层，因为即使是低延迟的系统，消息聚集也能带来性能提升. 结合短消息聚集技术，上述树形消息聚集结构应得到进一步改进.

3　消息合并

在调试并行程序时，多个进程执行的是同一个程序，根据预先定义的调试数据采集范围，结果数据里会有很多相同或相似的数据，但进程的执行进度不一定相同，因而不同进程下的结果数据差别很大，即一条命令发出后，返回的结果消息有可能出现多种情况. 本文将它们分为 4 种情况，根据 gdb/mi 接口的消息格式，设计了适合这 4 种情况的消息合并方法，最大程度地压缩消息量，同时兼顾系统的执行效率，最终使结果数据中保留 gdb 的原始输出内

容. 4 种情况及其消息合并方法如下.

(1) 完全相同的消息,指所有进程的返回消息完全相同. 在合并时,只需保留一条,其前端加上进程序号,以表示是哪些进程产生了这个消息.

(2) 部分相同的消息. gdb/mi 接口的消息一般由变量和值构成,这类消息是指 2 条消息中至少有一个值不同,所有的非值字符都相同. 处理时,将不同的值列出来,值之间用逗号分隔. 值前冠有中括号,表示值所产生的进程,括号中数值表示进程号,进程号之间用逗号分隔. 图 3 是对 3 个进程产生的消息进行合并的示例.

```
进程0:^done, bktp={number ="4", type="breakpoint",
disp="keep", enabled="y", addr="0x0804874 c",
func="main", file="hello_c.c", fullname=
"/home/zhx/hello_c.c", line="5", times="2"}

进程1:^done, bkpt={number ="4", type="breakpoint",
disp="keep", enabled="y", addr="0x0251334c",
func="main", file="hello_c.c", fullname=
"/home/zhx/hello_c.c", line="5", times="1"}

进程2:^done, bkpt={number ="2", type="breakpoint",
disp="keep", enabled="y", addr="0x0251334 c",
func="main", file="hello_c.c", fullname=
"/home/zhx/hello_c.c",  line="5", times="0"}

合并后:[0,1,2]^ done, bkpt={number ="[0,1]4,[2]2", type=
"breakpoint", disp="keep", enabled="y", addr=
"[0]0x0804874c,[1,2]0x0251334c", func="main", file=
"hello_c.c", fullname="/home/zhx/hello_c.c", line="5",
times="[0]2, [1]1, [2]0"}
```

图 3 3 个进程所产生消息的合并示例

(3) 不属于情况(1)、(2). 这类消息不能压缩,处理方法是在值前端加上所属进程的编号后再加到消息末尾.

(4) 一部分消息属于情况(1)、(2),另一部分属于情况(3). 这种情况虽然复杂,但是消息中有很多地方是可以压缩的,处理方法是前 3 种方法的组合.

4 实验分析

系统选取曙光 4000L 集群服务器中 18 个节点组成的并行计算环境,单个节点的配置为:Intel(R) Xeon(TM) CPU (3.00 GHz × 2),内存 2 GB, Broadcom BCM5721 1000Base-T 网卡, Linux 2.4 操作系统, OpenMPI 1.2.3 并行计算环境.

通过测试 2 个典型的调试命令的响应时间,比较改进前后树形结构的性能(所有的返回消息均做合并处理). 图 4 显示的是-gdb-version 命令的响应时间,可以看出改进后的树形在大多数情况下优于改进前的树形. 在某些情况下,进程的分配能充分利用节点的并行性,这时响应时间会变得很小,这就是图中进程数为 512 时出现凹陷的原因.

图 5 显示的是-break-insert 命令的响应时间,当进程数超过 512 时,改进后的树形优势明显.

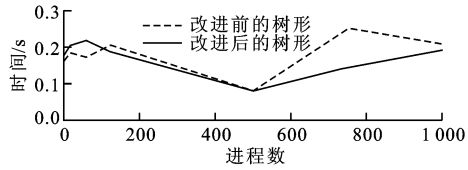

图 4 -gdb-version 命令在 2 种树形结构中的响应时间比较

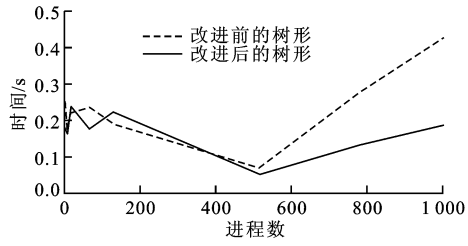

图 5 -break-insert 命令在 2 种树形结构中的响应时间比较

从图 5 可以看出,在少数情况下,例如在 128 个进程时,改进后的树形不如改进前的树形,原因可能有:改进后的树形层次数多于改进前的,命令的传送和结果的聚集都需要更多的 MPI 通信[8];本文测试环境是高速互连集群,节点间的通信延迟非常小,对于所测试的 2 个命令由于各个进程的输出相同,经过合并之后数据量变得很小,因此通信延迟不明显;与测试的进程数和节点数有关,少数情况下改进前的树形结构负载更加均衡. 尽管在少数情况下改进后的树形不如改进前的树形,但性能差距并不明显. 相比而言,-gdb-version 命令的返回时间比-break-insert 命令的返回时间要长,因而图 4 中改进后的优势比图 5 更显著. 在实际应用中,消息数据复杂而且数量大,可以预计的是,改进后的结构更具优势,更适用于网格等非纯高速互连的环境.

下面,在改进后的树形结构基础上采用了消息合并策略,消息合并前后系统的响应时间如图 6 所示. 可以看出:采用合并策略的响应时间保持在 0.2 s 左右,曲线很平缓并随进程数的增加变化不明显;未采用合并策略且当进程数较少时(8 个),由于系统不需要做消息的比较处理,所以整体表现比较好,但随着进程数的增加,响应速度会急剧下降,当有 128 个进程时,需要 33.2 s. 在调试大规模并行程序

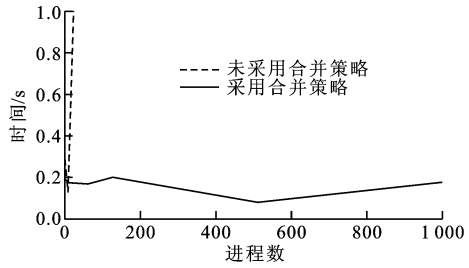

图 6　合并前后-gdb-version 命令的响应时间比较

时这样的响应延迟是不能忍受的.

-break-insert 命令正常的返回结果属于上述情况(2)，其需要比较及合并每一个变量和值，实际的响应时间基本保持在 0.2 s 以下(见图 7). 未采用合并策略、在进程数为 16 时，返回结果比较好，而当进程数继续增加时，响应时间很不稳定. 进程数超过 128 时，响应时间急剧增加，系统性能迅速下降.

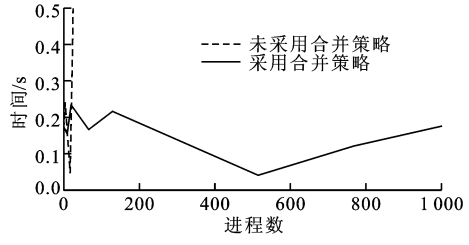

图 7　合并前后-break-insert 命令的响应时间比较

由此可见，消息合并策略的引入可以大幅度提高系统的性能.

5　结　论

本文提出了一种高效的新型消息聚集机制，采用了新的聚集架构并加入了消息合并策略，由此优化了大规模并行程序调试时结果数据的聚集算法. 应用该机制设计了一种能够支持远程调试并行应用程序的源码级调试系统框架，并实现了其原型系统. 该系统简化了传统的用户使用高性能计算中心计算资源的方式，屏蔽了各种复杂的硬件结构及处理流程，使得用户可以像调试本地程序一样调试远程集群服务器上的并行程序. 实验证明，新的消息聚集机制在调试大规模并行程序时可以减少调试命令的响应时间，提高远程调试系统的整体性能. 下一步工作将考虑系统支持底层并行环境的多样性及环境的异构性.

参考文献：

[1] LUMETTA S S, CULLER D E. The mantis parallel debugger [C]// Proceedings of the 1st Symposium on Parallel and Distributed Tools. New York, USA: ACM, 1996: 118-126.

[2] GRAG W. Parallel tools platform-parallel debugger [EB/OL]. [2008-08-12]. http://dev.eclipse.org/viewcvs/index.cgi/~checkout~/org.eclipse.ptp/doc/presentations/CDT_2006.pdf?cvsroot=Tools_Project.

[3] EDGAR G, GRAHAM E F, GEORGE B, et al. Open MPI: goals, concept, and design of a next generation MPI implementation [C]// Proceedings of 11th European PVM/MPI Users' Group Meeting. Budapest, Hungary: Springer, 2004: 97-104.

[4] 陈勇，何克东，陆在朝，等. 曙光机群系统并行调试器的设计与实现 [J]. 计算机工程，2004，30(9)：50-52.
CHEN Yong, HE Kedong, LU Zaichao, et al. Design and implementation of a parallel debugger for dawning cluster system [J]. Computer Engineering, 2004, 30(9): 50-52.

[5] HOOD R. The p2d2 project: building a portable distributed debugger [C]// Proceedings of ACM SIGMETRICS Symposium on Parallel and Distributed Tools. New York, USA: ACM, 1996: 127-136.

[6] BALLE S M, BRETT B R. Extending a traditional debugger to debug massively parallel applications [J]. Journal of Parallel and Distributed Computing, 2004, 64(5): 617-628.

[7] PHAM C, ALBRECHT C. Optimizing message aggregation for parallel simulation on high performance clusters [C]// The 7th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems. Piscataway, NJ, USA: IEEE, 1999:76.

[8] Message Passing Interface Forum. MPI2: a message passing interface standard [J]. International Journal of High Performance Computing Applications, 1998, 12(1/2):1-299.

(编辑　苗凌　赵大良)