基于应用层逻辑的分布式集群架构在生物计算中的应用

An Application Level Logic Based Architecture in Bioinformatic Computing Cluster

彭　彬　朱　勇　（浙江大学城市学院）
马天驰　（浙江大学计算机科学与技术学院）

摘要：本文提出一种新型的服务器集群构架，用于解决生物信息学计算中所使用的分布式集群系统的负载平衡问题。该结构基于任务的应用层逻辑对不同种类的任务进行分类，使得同类任务运行在固定的计算机子群上，从而回避了不同类型负载之间的比对问题，简化了负载平衡操作的难度。并且，此结构充分利用了计算机的时间局部性和空间局部性原理，提高了整个系统运行的效率。

关键词：生物信息学　任务描述　负载平衡　集群计算　分布式

1　概述

在计算机集群系统中，一个与总体性能有极大关系的因素就是负载平衡（Load balancing）。负载平衡指在集群环境中，通过什么样的逻辑来保证各个服务器（或处理器）的计算量与其自身性能之比尽量相等，从而在提高服务器利用率的基础上减少整体任务完成时间[1]。负载平衡在粒度上分为三种[2]。

（1）细粒度负载平衡：主要用于共享存储的MP机器；

（2）中粒度负载平衡：用于MPP和NOW，主要进行进程（Process）一级的调度；

（3）粗粒度负载平衡：用于NOW，主要为任务（Job）级管理。

在考虑到性价比的情况下，采用大量的小型计算机集群要比采用昂贵的多处理机并行大型计算机合算得多，故而人们希望尽量使用多个小型计算机来替代单个多处理器大型机。然而在这样的体系结构中，由于调度程序无法面对指令一级，就无法实现细粒度的负载平衡，只能够选择中粒度或者粗粒度。而对于粗粒度的负载平衡，在其系统设计中，就会在方法上与细粒度负载平衡有很大差距，从而面临着更多的问题。

对此，我们设计出一种体系结构。它将任务从逻辑上分类，再尝试着为同一类型的任务制定量化标准。这样可以简化任务的逻辑，从而简化对任务复杂度的描述工作。对于异构的服务器集群，我们也分成多个域，使得每个域上的服务器专门服务它们所擅长的工作，并且，在各域之间还能够通过调整服务器而非调整任务的方法实现动态的负载平衡，从而也回避了任务调整中对任务逻辑的描述。

2　任务负载的描述方法

目前对任务负载的描述一般采用两种方案。一种是基于网络应用层逻辑的，即在任务实际运行之前，甚至在算法设计时，就已经对任务逻辑进行分析，确定了任务参数与其复杂度的关系，从而能够准确地预测出任务的复杂度。另一种是统计方案。统计方案是采用某个任务先前执行若干次的时间长短作为样本库，从而通过数理统计的途径来对当前尚未执行的任务之复杂性进行预测。这种方案的优点是可以不用人为地去关注任务的内部逻辑，而是在集群系统中进行自动估测。从借助UNIX中资源估测的方法出发，人们对这种方案的研究长达20年之久，在此期间不断对统计和预测的方法进行改进[4]。但根据统计理论的原则，一项统计估测终究只适用于方差比较小的样本库。对于一个分布均匀的样本库来讲，这样的预测是没有意义的。

一种折中的办法是将以上两种方案结合使用，即对于可预测的任务，使用统计方案，而对于其他任务则使用基于上层逻辑的方案，这样产生的问题是如何将两种方案结合起来。由于这两种方案的任务复杂度量

化标准完全不同，并且，即使是在基于应用层逻辑的方案中，不同类型的任务可能也会有不同的负载描述方法。它们可能是完全异构的，中间不存在任何运算和比较的方法。这样就意味着一个服务器集群只能处理单一的任务类型，如果同时存在着多种类型的任务，就需要经过一个很繁琐的任务负载换算过程。然而在目前网络计算模式下的应用中，往往一个服务器集群要负担对多个软件提供者进行硬件承载任务。如果我们要求一个集群只能处理一种类型的任务，这显然是不合理的。针对这种情况，我们提出了以下的体系结构。

3　服务器集群基本框架

图1是此结构的基本框架。

该体系结构有以下优点：

(1) 用分组的方法，将同类的任务放置在一起处理，这样回避了在不同逻辑之间进行复杂度比较的过程，使得每个负载平衡代理的任务单一化，大大提高了负载平衡的效率和准确度。

(2) 同类的任务在逻辑上和数据访问上必有相似和相关之处。用固定的服务器执行固定类型的任务，实际上是利用了时间局部性和空间局部性原理，使得缓存在服务器上的数据可以反复被利用，从而提高服务器执行的效率。

4　动态负载平衡算法

以上介绍了将一个集群划分成多个的子群，分别负责不同类型任务的体系结构。需要指出的是，该结构是一个动态的框架，域和域之间存在着动态的负载平衡，这样才能够充分利用到集群中服务器之间互补和并行计算的特性。下面介绍在这样的体系结构下的动态负载平衡算法。

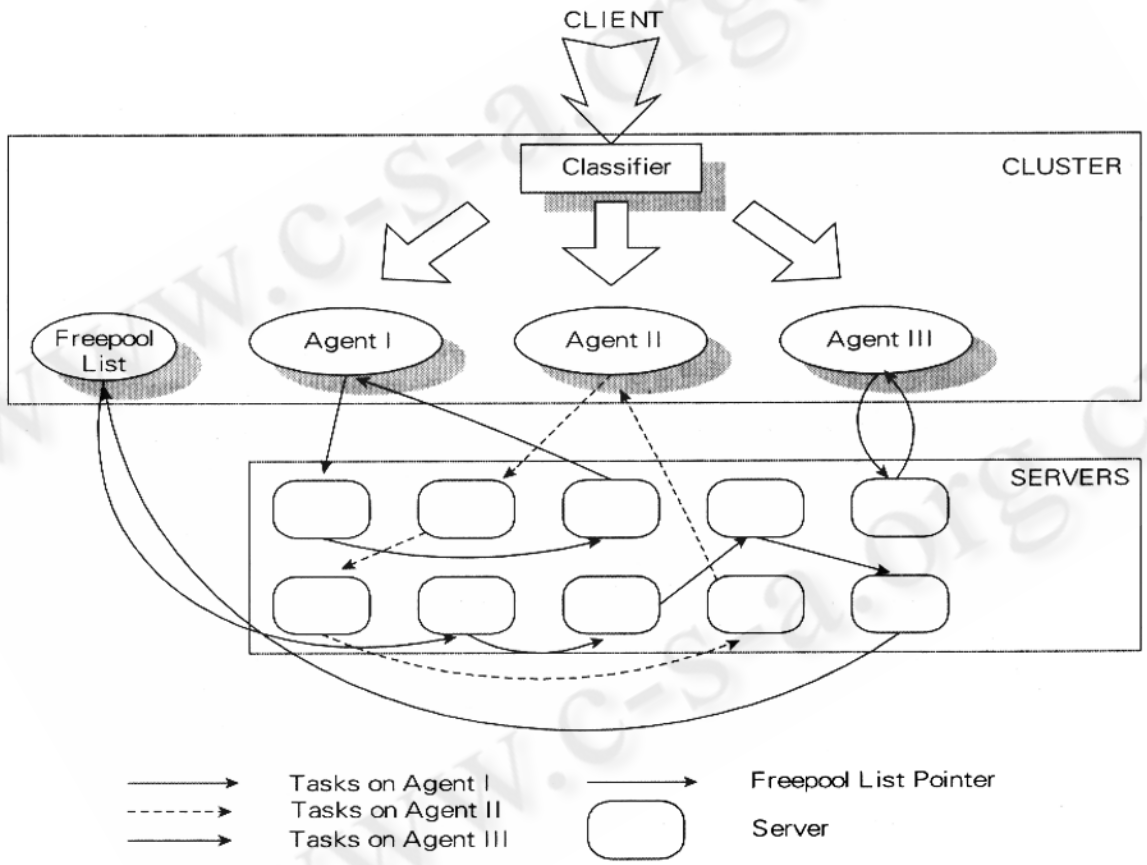

图 1　整体结构

通常意义上的服务器间动态负载平衡运行步骤如下：首先评估各服务器的负载，当某些服务器负载过高时，即令集群控制器从高负载服务器的任务队列中抽

取一些任务，置入低负载服务器任务队列。这样的方法的缺点是任务移动界面复杂。任务的移动过程需要涉及到任务的界面和元数据（metadata），而任务的描述和解释都需要经过处理（查找，计算），且没有统一的标准，这样就增加了任务移动的难度。尤其对于某些复杂的任务，这样的处理可能要消耗大量的时间。并且，对于本文中提出的体系结构，域和域之间的任务调整是不允许的，因为这样会造成各域之间任务逻辑混乱。如果造成了这种混乱，就丧失了分域带来的好处，使得调度服务器又需要重新面对任务逻辑的描述。

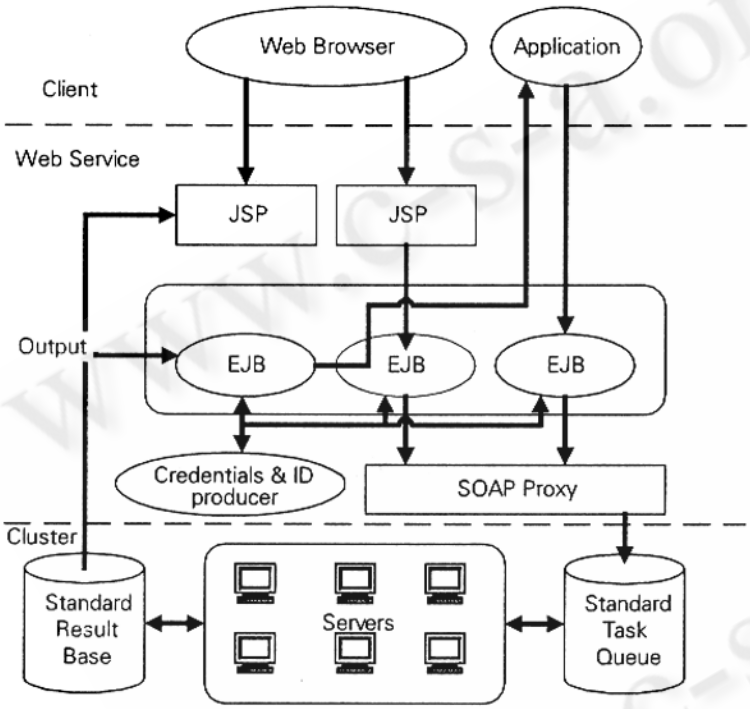

图 2　BLC 整体构架

为了不破坏每个域的逻辑完整性，我们的做法是由低负载域选择出一台空闲服务器调入高负载域。这样的负载平衡在界面上要比移动任务简单。因为对于一台普通服务器来说，描述它所使用的参数是统一的，也就是可计算的。

我们使用空闲池的方法来进行动态的服务器调整。在某种任务特别过剩，而其他一些任务缺乏的情况下，可以动态调整低负载域的服务器到高负载域中。在 Server 集群中设立一个空闲池（Server Pool），服务器过剩的域可以直接将剩余的机器扔到空闲池中，而任务过剩的域则直接向空闲池索要空闲服务器。

每个负载均衡服务器定时发布自己域的负载信息，这样每个负载均衡服务器中都会维护一张所有域负载的列表，并不断更新。如果某些域负载过高，就会请求分配计算机。空闲池接到广播后，首先检查池中是否存在着空闲的机器（一般不会，除非在系统刚刚开始运行或者任务本身比较稀少的情况下），如果不存在空闲的机器，空闲池会向调度服务器发送需要机器的信息。这时，调度服务器会向所有的域代理广播此信息。而各个域代理根据自己与平均负载的差距来决定是否要调整自己的一些服务器去空闲池。此方法的关键在于由各个域自行决定是否进行调配，而非统一控制。这样做就形成了域和域之间的异步通讯，从而使得系统成功地回避了在各个域负载平衡服务器以不同逻辑计算出的复杂度之间进行无意义的比较。

5　生物计算集群的实际架构

本文所提出的技术被应用于构建生物计算的集群系统，收集和改进了大量专业的基因序列处理软件，并对外以 Web Service 的形式提供计算服务。整套系统基于 J2EE 构架，用户可以在应用程序中直接调用远程的 EJB 组件，也可以通过 BLC 外部提供的 JSP 页面进行任务提交，不久的将来还将提供标准 XML 格式的任务提交方式。在任务提交后，将交由认证代理进行权限认证，并给出相应的信任证（Credential，将用于远程数据库或资源的授权访问，目前此功能尚未实现）或者流水序列号（ID）。对于通过认证的任务，用户将得到该任务的序列号。

如图 2 所示，整个 BLC 构架分为三层，分别是：

（1）用户层（Client）。这一层位于用户的本地端，支持 Web 方式的调用与应用程序的直接调用。

（2）服务层（Web Service）。这一层包括 JSP 网页和 EJB 组件，它们起到连接服务端和客户端的作用，并将运行在服务器平台上的软件集成在 J2EE 环境中。

（3）集群层（Cluster）。这一层为上文所重点介绍的结构，其功能是高效地完成服务层所交给的任务，并将结果返回。

用户提交的任务将通过一个支持 SOAP 标准的代理（proxy）转换成 XML 格式，并存放在标准任务队列

(Standard Task Queue)中。服务器集群从该队列中不断读入新的任务,并按照前文所介绍的负载平衡策略将任务分配给相应的服务器。服务器在完成任务后,将结果置入标准结果库(Standard Result Base)中,以等待服务层的查询。用户可以凭借提交任务时所得到的序列号访问相应的查询组件,从而对任务的完成状态(如是否完成,已运行时间,是否出现异常等)以及最终结果进行查询。

6 测试与结论

首先,我们对 BLAST 软件的运行情况进行测试。我们选取两种类型的 BLAST,分别是 Nucleotide – nucleotide search (blastn) 和 Translated – nucleotide – protein(blastx)。根据 Sun 公司对常用生物软件性能的评估分析,在 BLAST 中,blastn 的 CPU 主频加速比比较高,而 blastx 的 CPU 数量扩展加速比比较高(见图 3)。这样,我们尽量将双 CPU 的工作站分给 blastx 程序。

对于核苷酸数据库(nt)大小的选择是可能影响到系统性能的重要因素(工作站的内存为 1G)。即如果库小于单个工作站的内存大小,则有可能发挥出工作站内存的缓存作用。我们分别使用两种大小的 nt 库(0.8G/1.3G)进行测试。此外,每次任务提交的序列文件中所包含的序列数量也对性能有所影响。对于大库和小库,我们都分别用短序列文件(包含 1 条序列)与长序列文件(包含 5 条序列)进行测试,这样一共需要进行 4 次测试。每次测试中,我们将 50 个 blastn 和 50 个 blastx 的任务置入标准任务队列中去,并在执行后根据日志查询并记录全部任务完成 10% ,20% ,... , 100% 时所对应的时间。表 1 为具体测试序列和库的长度数据。

我们先观察动态负载平衡的运行情况。发现随着任务的比例改变,各域的服务器数会随之动态调整。当开始执行时,大量 blastx 程序涌入并抢占集群中的服务器。而在最后阶段,blastx 域中的任务率先完成,这时服务器逐渐从 blastx 域转向 blastn 域,最终全部投入到 blastn 的执行中。此实验验证了本文中提出通过动态调整服务器进行负载平衡算法的正确性。

接下来,我们比较本文负载平衡方法与一般负载平衡算法的性能。图 3 中记录了四次测试的比对数据。其中 A 线为不分域的负载平衡,其使用平均算法,即总是将新任务分配给负载最低的机器执行。而 B 线为本文提出的多子域负载平衡,在域内使用平均算法。

表 1　测试材料基本数据

程序	短序列总长度(Byte)	长序列总长度(Byte)	短数据库序列数	短数据库总长度(Byte)	长数据库序列数	长数据库总长度(Byte)
blastn	31,616	151,502	842,981	3,315,872,547	1,170,581	4,968,036,787
blastx	31,616	151,502	888,128	278,103,739	888,128	278,103,739

我们可以看出,在选择大库/长序列文件的情况下,两种负载平衡方案执行时间相差不多。算法 A 略快,这主要是因为服务器性能的偏向。不过由于 blastx 域的数据库小,服务器性能又高,所以很快执行完毕。这时所有的机器都调入 blastn 域中,此后由于单域执行,所以并不能表现出性能的优越性。

而在选择小库/短序列的情况下,分域负载平衡算法表现出非常优越的性能。这说明了服务器集群对于小于内存的数据库有着良好的缓存功能,而各域中任务执行的单一性使得可以充分利用这样的缓存。在实际应用中,我们将大数据库分成若干个小库,放在多个域中依次进行比对,从而解决了大数据库无法进行缓存的问题。

最后,我们将该集群的性能与曙光机上的 3 个节点进行对比。下表为硬件比较:

表 2　硬件比较(占优方加黑)

硬件指标	工作站集群	曙光服务器
CPU 主频	850MHz	400MHz
CPU 总数	4 * 2 + 2 * 1 = 10	3 * 4 = 12
内存	4 * 1G + 2 * 256M = 4.5G	3 * 1G = 3G
外存	普通硬盘	SCSI 磁盘阵列(空闲)
网络	100M 网卡	650M 高速
硬件成本	50000RMB	3000000RMB

大库长序列测试

算法	所用时间(s)									
	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
平均算法负载平衡	768	1013	1432	1554	1702	2163	2680	2989	3122	3341
多子域负载平衡	372	628	893	1186	1424	1948	2329	2615	2796	3021

大库短序列测试

算法	所用时间(s)									
	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
平均算法负载平衡	205	333	387	442	526	555	601	702	795	902
多子域负载平衡	92	153	201	250	280	340	389	557	595	650

小库长序列测试

算法	所用时间(s)									
	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
平均算法负载平衡	292	463	604	827	1128	1354	1609	1871	2292	2743
多子域负载平衡	340	429	545	701	870	1057	1211	1370	1706	2056

小库短序列测试

算法	所用时间(s)									
	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
平均算法负载平衡	108	162	235	306	379	465	539	563	635	739
多子域负载平衡	90	111	145	175	226	274	285	330	373	424

图 3 负载平衡算法之间的性能比对

综上所述,本文提出的多子域分布式集群系统动态负载平衡算法,成功地回避了任务复杂度描述,任务迁移等在一般负载平衡算法中的难点问题,并能够充分利用各服务器本机的缓存机制,较大地提高了集群系统的执行效率。

参考文献

1 Albert Y. Zomaya, Yee – Hwei Teh, "Observations on Using Genetic Algorithms for Dynamic Load – Balancing", IEEE Trans. on Parallel and Distributed Systems, Vol 12, No. 9, pp. 899 – 911, Sept. 2001.

2 郑纬民、杨广文、周佳祥,"工作站/PC 群集计算机系统",清华大学计算机科学与技术系.

3 Devarakonda, M. V., Iyer, R. K., "Predictability of process resource usage: a measurement – based study on UNIX", Software Engineering, IEEE Transactions on, Volume: 15 Issue: 12, Dec. 1989, Page(s): 1579 – 1586.

4 W. Leland and T. Ott, "Load – balancing heuristics and process behavior", in Proc. Performance '86 and ACM SIGMETRICS Conf., Raleigh, NC, May 1986.