基于批处理补丁的流媒体后缀动态缓存算法*)

蔡青松[1] 李子木[2] 覃少华[1] 胡建平[1]

(北京航空航天大学计算机科学与工程系 北京100083)[1] (清华大学网络中心 北京100084)[2]

摘 要 在IP网络上高效传输流式存储型A/V数据是实现诸如VOD等应用的基础。当前一些典型的传输方案考虑了服务器调度策略以降低骨干网带宽消耗和服务器负载,但并未考虑媒体后缀的缓存策略。本文在带前缀的OBP算法基础上提出了流媒体对象后缀的增量式缓存及快速释放算法ICBR,并推导出了采用IC算法所需的骨干网带宽的理论结果。通过针对上述两种算法的仿真实验,本文的结果表明:即使在有限的缓存容量的前提下,采用IC算法和ICBR算法对媒体对象的后缀进行动态缓存可以显著降低骨干网链路上传输的补丁数据量,其骨干网带宽消耗显著优于OBP,从而在保证客户端较小的播放启动延迟的情况下有效降低了流媒体传输中骨干网带宽的消耗和服务器的负载。

关键词 CDN,流媒体,缓存代理,批处理补丁,前缀缓存,后缀缓存

A Dynamic Cache Algorithm of Media Suffix Based on Batch Patching

CAI Qing-Song[1] LI Zi-Mu[2] QIN Shao-Hua[1] HU Jian-Ping[1]

(Department of Computer Science and Engineering, Beijing University of Aeronautics and Astronautics, Beijing 100083)[1]

(CERNET Center, Tsinghua University, Beijing 100084)[2]

Abstract It is the basis for types of applications such as VOD to deliver stored media files over IP-based network effectively. Current researches consider the server schedule schemes to decrease the consumption of backbone bandwidth and the server load while not considered the cache policy of media suffix. This paper proposes an Incremental Caching Burst Release (ICBR) algorithm of media suffix that works upon the Optimized Batch Patching scheme, the theoretical expression of the aggregate transmission rate over the backbone network is derived when using the incremental caching algorithm. The results of simulation show that, even the cache capacity is limited, our algorithm can remarkably reduce the patching traffic of the backbone, thus outperforms OBP that with prefix cached by reducing the consumption of backbone bandwidth and the load of the original server while still have low startup latency of client side playback.

Keywords CDN, Streaming media, Cache proxy, Batch patching, Prefix caching, Suffix caching

1 引言

近年来,随着网络技术的进步和网络基础设施的不断完善,许多面向Internet的应用得到了迅猛的发展,如大型运动会(奥运会等),大型博览会的信息发布和查询,视频点播(Video On Demand),远程教育及在线视频关播等。然而,这些应用的增多和网络规模的日益扩大导致了网络上传输的内容的急剧增加,使得有限的网络资源逐渐变得紧张。从整个网络的主要构成看,骨干网带宽资源目前已经成为制约Internet应用发展的瓶颈。在这种情况下,边缘服务(Edge Services)的出现一定程度上缓解了上述问题。内容分送网络CDN(Content Delivery Network)则是为加速Internet内容传输的一种集成边缘服务的应用层逻辑网络。

多种宽带接入技术(如ADSL,HFC,Wireless LAN等)的应用一方面为用户提供了更高更稳定的接入带宽,另一方面,它更加剧了骨干网带宽资源和服务器资源的消耗。数字视频技术的不断成熟使得如VOD等应用大量出现,这些应用所使用的存储A/V数据在传送时需要消耗大量的网络带宽资源和服务器I/O及存储资源。虽然流式媒体(以下简称流媒体)压缩技术一定程度上使得这些数据在Internet上传输成为了可能,但是流媒体数据的本身固有的特性使得其在Internet上高效传输异常困难。目前,研究如何在现有的网络构架下高效地在Internet上传输流媒体资源已经成为一个普遍关注的热点课题。国内外已有的研究和实践结果表明,以下技术对于解决多媒体资源在Internet上的高效传输具有重要意义:1)组播技术;2)缓存代理。

组播技术通过在组播分支点将包进行复制,然后沿不同的路径将包发送到同组的各个用户,而发起组播的服务器只需要将包的一个备份发送出去即可,这与单播的情况相比大大降低了网络的带宽消耗。然而,由于现有组播协议在安全性、可扩展性和可管理性等方面存在着缺陷以及支持组播的网络并没有进行广泛的部署,因此单纯依赖组播技术在Internet上传输流媒体数据是不实际的。

缓存技术通过将用户频繁请求的内容复制和缓存在离用户最近的接入点部署的缓存代理服务器上(以下简称缓存代理),并通过一定的策略维持与源服务器的一致性更新,从而使得一些被频繁访问的资源可以直接从缓存代理获取而无须再请求源服务器,这样骨干网络的带宽和服务器资源的消耗都得到明显的降低。另外,由于缓存代理是离用户最近的访问接入点,用户请求的响应速度也得到了显著的提高。然而由于多媒体对象普遍具有较大的文件尺寸和较长的播放持续时间,传统的缓存WWW资源的方式并不适用于流媒体对象的

*)国家自然科学基金资助项目(基金编号:60103005,基于Internet的高速CDN关键技术研究)。蔡青松 博士研究生,研究方向:网络与多媒体,分布式计算。李子木 博士,副教授,研究方向:分布式多媒体数据库,网络体系结构,多媒体传输。覃少华 博士研究生,研究方向:网络与多媒体,分布式计算。胡建平 教授,博士生导师,主要研究领域:高性能分布与并行计算,计算机体系结构,网络多媒体。

缓存。也就是说，在有限的缓存容量的情况下，全部缓存整个流媒体资源会导致缓存代理只能容纳少数流媒体资源，从而导致缓存命中率的急剧下降。当缓存服务器需要替换掉某个缓存的对象时，它往往正处在为某个请求服务的状态，这就使得替换不能够正常进行，影响了缓存代理的效率。典型的具有缓存代理的流媒体分送系统的体系结构如图1所示。

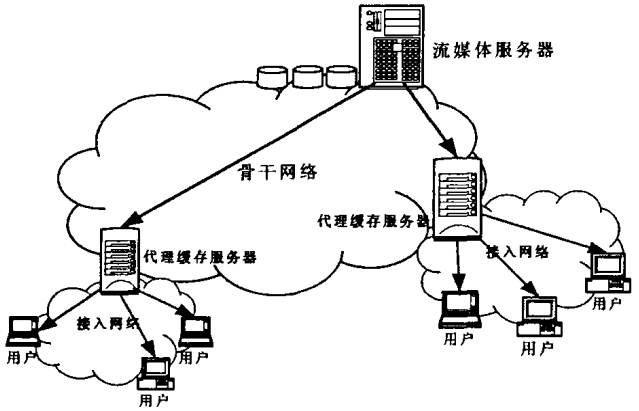


图1　典型的流媒体分送体系结构

无论是组播技术还是代理缓存技术，在传输流媒体数据时都需要解决如下三个主要问题：1）骨干网带宽消耗；2）服务器负载；3）客户端的播放延迟和 QoS（这里我们将客户端的 QoS 表现为播放延迟和丢包率）。本文结合批处理补丁（Batching Patching）技术从上述三个方面分析了它们的性能，在此基础上本文并提出了流媒体后缀动态增量式缓存及释放算法 ICBR。分析和实验结果表明，这种方式在保持客户端具有较小的播放启动延迟的情况下，大大降低了由于请求补丁数据而导致的骨干网链路上的数据流量，从而降低了骨干网带宽资源消耗和服务器负载，同时也相应地提高了客户端 QoS，其结果显著优于现有方法。

2　相关工作

采用基于缓存复制协议的内容分送网络（CDN）对于 WWW 对象的有效缓存显著降低了骨干网络的带宽需求和 Web 服务器负载。然而由于流媒体对象本身固有的特点如具有较长播放持续时间（几分钟至数十分钟甚至几小时），较大的文件尺寸（数十兆至几百兆），播放的顺序性，及丢包敏感性等特性，使得传统的 Web 对象缓存协议不适用于在 Internet 上传输流媒体对象。研究如何在 Internet 上高效传输流媒体对象的工作当前主要集中于研究应用层组播协议和缓存算法。

与为每个用户请求都建立一个单播通道相比，通过组播方式传送流媒体数据降低了网络带宽消耗和服务器的负载。文[1,2]提出了周期广播策略来利用组播的优点加速流媒体传输，它在服务器上将每个视频文件划分为多个部分，然后对于每个部分在多个通道上不断广播，这种方法被证明只有在用户的请求到达速率较高时才能体现出一定的效率。文[3]采用批处理的方式来处理用户请求：在一个批处理间隔内到达的请求都被服务器累计起来并且统一在每个间隔结束的地方进行服务。这种思想虽然节省了服务器的带宽和 I/O 资源，但是却增大了客户端的播放延迟，导致了更低的 QoS。比简单批处理方法更具有带宽效率的方法是补丁技术[4,5]：每次处理由正常通道 RC（Regular Channel）单独进行服务或者正常通道与补丁通道 PC（Patching Channel）联合进行服务。其中 RC 传送整个流媒体对象，而 PC 则传输请求到达并且加入 RC 时缺失的流数据部分。客户端同时接收 RC 和 PC，即播放 PC 同时缓冲 RC，当 PC 播放完时对缓冲区中的内容以 FIFO 方式播放。显然，补丁技术通过发送同一个流的补丁减少了 RC 数量，但它却增加了 PC 流的数量，使得服务器的外出链路上的所需通道个数并不能得到显著降低。当请求到达速率较高时，甚至会多于 RC 个数。再者，这种方案需要客户端具有较大的缓冲区。因此，虽然补丁技术获得了骨干带宽的节省，却没有对服务器的负载进行优化，同时客户端资源需求也较大。文[6]提出了前缀缓存的思想，通过前缀缓存，优化了客户端的播放启动延迟。文[10]提出了对补丁进行周期调度策略。文[11]对文[6]进行了扩展并提出了改进，即对流媒体进行固定前缀缓存，同时对补丁进行批处理同时结合流媒体服务器调度和缓存代理来服务客户。文[11]提出了最优批处理补丁策略 OBP 并得出了最优补丁窗口，它通过设置不同的批处理间隔建立了一种用户可控制的可选服务类型和服务质量以获得延迟和使用费用之间的折中。

另外一个有效传输流媒体对象的研究热点是对流数据的缓存和替换策略。对流媒体对象的缓存策略目前有：1）整体缓存；2）部分缓存；如前所述，策略1）使得缓存的命中率很低，导致较低的缓存效率。策略2）部分缓存可以提高命中率，但是却带来了需要请求同一流对象的其它部分的额外开销。因此，在这两种策略中，采用策略2）并仔细考虑如何降低对同一流对象其它部分请求的开销则可以获得较好的缓存效率。文[6～9]分别基于不同的部分缓存策略进行流媒体传送。研究工作[6,9]表明前缀缓存可在很大程度上缓解客户端播放启动延迟大的问题。对于后缀，在缓存容量有限的情况下，采取适当的替换算法动态缓存和替换缓存中的内容则可以获得更高的效率。文[7]比较了基于资源的缓存 RBC 与 LFU 算法，提出了结合 LFU 和间隔缓存的缓存策略。文[8]提出了静态缓存，动态缓存和多缓存协作，并提出了一个自组织多缓存协作体系结构 SOCCER 来缓存流媒体数据。

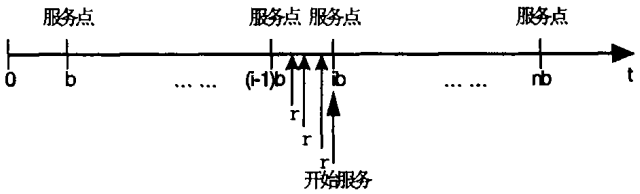
事实上，由于流媒体普遍具有较大文件的长度，在有限的缓存容量的前提下仅依赖于缓存来降低骨干网带宽资源消耗和服务器负载是不够的。而辅以前缀缓存和有效的后缀缓存以及批处理补丁等应用层多播协议则可以很大程度上缓解上述问题，同时也能有效地降低客户端播放延迟。文[12]结合了服务器调度和代理缓存策略分析了批处理补丁辅以前缀缓存的效率，性能评价结果优于简单批处理，补丁和批处理补丁等方案。

无论是周期广播，批处理，补丁，批处理补丁还是前缀缓存等策略及联合服务策略，均没有考虑对流媒体后缀的动态缓存，而流媒体对象的后缀的有效缓存可以大大降低骨干网络带宽消耗。本文分析了带前缀缓存的批处理补丁方案的性能，提出了后缀增量式缓存算法和释放策略并对缓存算法进行了形式化推导，给出了仿真结果及性能评价。性能评价结果指出即使在有限的缓存容量下，我们的缓存及释放策略均优于带固定前缀的批处理补丁策略，大大降低了因获取补丁而在导致的骨干网流量，从而进一步降低了骨干网带宽资源的消耗和服务器的负载。

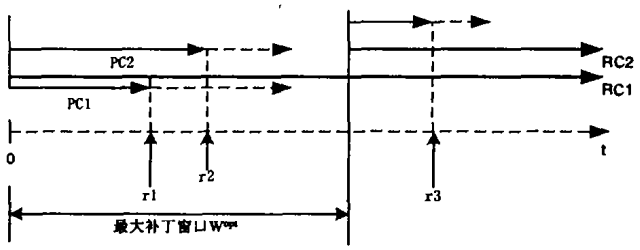
3　问题分析

本节从流媒体分送系统的骨干网带宽资源消耗，服务器负载（这里指服务器在传输一个流对象期间需要建立的通道个数）和客户播放启动延迟等几个方面分析了简单批处理，补丁和批处理补丁等几种方案的性能。在下面的分析中，若采用了部分缓存，则我们把一个流媒体对象的固定缓存长度为 b

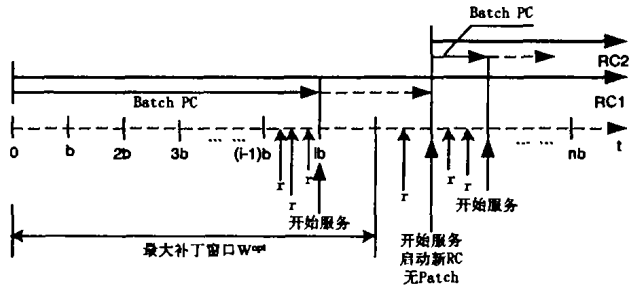
的前部分数据称为流媒体前缀(简称前缀),而把后面部分的数据称为流媒体后缀(简称后缀)。这里假定当前流媒体对象播放持续时间为 T,用户的请求流符合到达速率为 λ 的泊松分布,为简单起见,暂不考虑线路传输延迟。



a) Batch



b) Patch



c) Batch Patching

图2 Batch,Patch 和 Batch Patching

3.1 批处理

批处理如图2.a)所示。服务器将其调度时刻按照间隔 b 进行划分(设 $T=nb$),设 $P=e^{-\lambda b}$ 为 b 间隔内无请求到达的概率,对于所有在区间$[(i-1)b,ib](i\in Z^+)$到达的所有请求都会在时刻启动一个新的通道来为其服务。

对于同一流对象的请求,采用简单批处理时服务器需要建立的平均通道个数(C_{avg})为:

$$C_{avg}=\sum_{i=1}^{n}(1-P)=n(1-P) \tag{1}$$

每个服务器通道均按照媒体对象的播放速率 r 传输流媒体数据,则骨干网络上平均所需带宽(B_{avg})为:

$$B_{avg}=C_{avg}r=n(1-p)r \tag{2}$$

对于任意一个到达的请求,其在最近到来的批处理时刻会得到服务。因此客户端的平均延迟(D_{avg})为:

$$D_{avg}=\frac{b}{2} \tag{3}$$

显然,批处理调度策略虽然有效降低了骨干网络带宽消耗和服务器负载。但式(3)表明,当批处理间隔较大时,客户端具有较大的播放启动延迟。

3.2 补丁

补丁的过程如图2.b)所示。在同一个补丁窗口 W_{opt}[11]内,所有的请求都共享一个 RC 流。对于在同一个补丁窗口内的不同时刻到达的请求,各自需要请求不同长度的 PC。在一个 W_{opt} 内,服务器所需建立的平均通道个数为:

$$C_{avg}=\lambda W_{opt}+1 \tag{4}$$

因此通道个数取决于请求到达速率 λ。当对该流对象的访问"热度"较高时,服务器所需的通道个数呈线性增长,而除了一个通道为 RC 流使用之外,其它所有的服务器通道均为 PC 流所导致。

根据(4)式,在一个补丁窗口内到达的请求个数等于需要建立的 PC 个数,即 λW_{opt}。设 PC_{avg} 为平均所需的补丁长度,则:

$$PC_{avg}=\frac{W_{opt}}{2}$$

因此,骨干网所需的平均带宽 B_{avg} 为:

$$B_{avg}=\frac{(\lambda W_{opt}*PC_{avg}+1)r}{W_{opt}}=\frac{\lambda W_{opt}r}{2}+\frac{r}{W_{opt}} \tag{5}$$

补丁策略在请求到达速率较小时可以有效地降低服务器通道数量和骨干网带宽流量,并且在请求到达时立即提供服务,具有较小的服务延迟。但是,这种方法需要客户端在播放 PC 时缓冲到来的 RC 数据,因此需要具有较大的缓冲区,同时客户端也必须具备同时从两个通道接收流数据的能力。

3.3 最优批处理补丁(Batch Patching)与带前缀静态缓存的最优批处理补丁(Optimized Batch Patching-OBP)

如图1.c)所示。批处理补丁(Batch Patching)技术使所有到达客户端的流数据都流经图1所示的缓存代理,并且结合了简单批处理和补丁技术的特点,在间隔点对当前间隔内到达的请求加入到 RC 流并且同时获取补丁。这种方法减少了所增加的因获取 PC 流而导致的服务器所需建立的通道数量,同时也使骨干网带宽消耗得到降低。按照前面的假定,可计算出服务器需要建立的平均通道数量:

$$C_{avg}=1+\sum_{i=1}^{\lfloor\frac{W_{opt}}{b}\rfloor}(1-P)=1+\lfloor\frac{W_{opt}}{b}\rfloor(1-P) \tag{6}$$

所需的平均骨干网带宽为[8]:

$$B_{avg}=\frac{(1-P)W_{opt}^2r+(1-P)bW_{opt}r+2bTr}{2bW_{opt}+\frac{2b^2}{(1-P)}} \tag{7}$$

批处理补丁技术有效地降低了服务器负载和骨干网带宽资源消耗。但是每个用户的请求需要等待到整数倍 b 的时间点才能得到服务,因此,其服务延迟等于简单批处理的情形。具有固定前缀 b 的最优批处理补丁 OBP 不仅解决了启动延迟的问题,而且优化了批处理补丁的补丁窗口。其算法结合了前缀缓存、批处理调度和补丁技术,在补丁窗口内的每个批处理点将当前批处理间隔内所有到达的请求统一加入到 RC 中并统一获取补丁数据,当调度时刻超过最优补丁窗口 W_{opt} 时,下一个到达的请求将导致一个新的 RC 启动(其播放长度为 $T-b$),其后的第一个批处理间隔建立同样大小的补丁窗口 W_{opt} 为最近启动的 RC 提供补丁服务,依此类推。假定请求在 t_q 时刻到达,代理立即使用缓存的前缀为其服务,由于 $t_q\in(t_{k-1},t_k]$,$k\in Z^+$ 且 $k>1$,因此具有 b 前缀的缓存隐藏了由于批处理而导致的用户端播放延迟。具有固定前缀 b 的 OBP 算法中服务器所需建立的平均通道个数等于式(6)。所需骨干平均带宽为:

$$B_{avg}=\frac{(1-P)W_{opt}^2r+(1-p)bW_{opt}r+2b(T-b)r}{2bW_{opt}+\frac{2b^2}{(1-P)}} \tag{8}$$

OBP 不仅保持了批处理补丁技术的服务器端性能和骨干网带宽效率,同时也保持了客户端的低延迟特性。但在 OBP 中客户端必须具备能同时接受来自三个通道的流的能力:由于 $t_k-t_q<b$,即客户端在播放完成前缀 b 之前必须在 t_k 时刻加入 RC 并且请求 PC。同时客户端也必须具备能容纳

$W_{opt}+b$ 播放时间长度的流媒体数据。

通过以上分析知道，OBP 结合了批处理调度，补丁和前缀缓存技术，有效地降低了骨干网带宽消耗和服务器负载及客户端启动延迟。但是通过分析不难得出，除前缀外，占媒体数据流量绝大部分的媒体对象后缀在按照 OBP 方案进行补丁数据传输时会存在重复的数据流量，这仍将会增大骨干网带宽资源消耗和服务器负载。下面我们提出媒体后缀的增量式缓存算法来降低这些重复的补丁数据流量，在保持较低的客户端启动延迟前提下，进一步降低骨干网带宽消耗和降低服务器负载。

4 批处理补丁的后缀增量式缓存和快速释放算法 ICBR

从前面的分析得知，批处理补丁技术通过正常通道 RC 和补丁通道 PC 来服务用户的请求，同时将补丁进行批处理和对固定长度的前缀进行缓存。这种方法大大降低了服务器负载和骨干网带宽消耗，同时也使客户端的播放启动延迟得到最大程度的降低。但是这种方案有以下几个缺点：首先，客户端必须具备能同时从多个通道接受流数据的能力。其次，当客户端完成对前缀数据的播放开始播放补丁数据时，它也必须具有充分大的缓冲区缓冲到来的 RC 数据，设 Buf 为缓冲区大小，则 $Buf \geqslant L_{0,ib}$，其中 ib 为其最近的批处理执行时刻，$L_{0,ib}$ 为该时刻启动的补丁流长度。第三，每次批处理所请求的补丁数据都是在时间点 b 开始从服务器获取，这使得骨干网链路上存在大量重复的数据流量，导致骨干带宽消耗和服务器负载较大。

由于代理是离用户端最近的接入点，将流媒体后缀的补丁在缓存代理中进行动态缓存的策略不仅可以进一步降低骨干链路上的流量，而且也能提高用户的访问速度。同时由于补丁数据缓存在缓存代理中，客户端可以播放完前缀再播放来自缓存代理中的补丁数据，而无须在最近的批处理点播放前缀的同时接收 PC 数据，减少了客户端需并发接收流的通道个数。根据流媒体数据播放时具有顺序性的特点可知，较早时期的补丁数据可以与同一窗口内较晚时期的补丁共用。因此在本节中我们利用在缓存中固定缓存长度为 b 的前缀的同时，提出了对后缀补丁数据的缓存和释放策略。通过后缀补丁数据的缓存，骨干网链路的流量得到了明显的降低，从而减轻了骨干资源和服务器资源的消耗。

由于我们在代理中固定缓存了长度为 b 的前缀，因此，在没有特别指出时，后缀均指长度 $T-b$ 为的流媒体数据，并且下面均从与前缀 b 的相对偏移即从0开始计算(也就是说，后缀中$[0,b]$的数据实际是流媒体中$[b,2b]$的数据，由于一般情形下，$T/b>>0$，因此这样是合理的)。

4.1 增量式缓存算法 IC

假定缓存容量大于窗口大小。如果在$[0,b]$有请求到达，则在 b 时刻缓存代理需要从服务器获取长度为 b 的补丁来服务该请求，同时将该长度为 b 的数据缓存起来。如果在$[b,2b]$也有请求到达，则在$2b$ 时刻缓存代理从服务器获取长度为$2b$ 的补丁，但是由于$[0,b]$的补丁已经存在于缓存中，因此此时只需要从服务器获取长度为 b 的$[b,2b]$的补丁即可，如果无请求到达，则不做处理。依此类推。

如前，客户请求流按照泊松分布到达。定义 $P=e^{-\lambda b}$ 为在一个批处理间隔内无请求到达的概率，其中 λ 为请求到达速率。为分析简单起见，使补丁窗口大小等于批处理间隔的整数倍，即 $Nb(N\in Z^+)$。设$\{X_1,X_2,\cdots,X_N\}$为 N 个批处理间隔内请求到达的随机变量($X_i=0,1,\cdots,n\ n\in Z^+\cup\{0\}$)，显然，只要 $X_i>0(i\in[1,N]$为整数)，在 ib 就需要产生一次批处理补丁，其长度为 iP。根据泊松分布的间隔独立性及各间隔内分布的同分布性，我们得到：

$$\begin{aligned}P_i &= P(X_1\neq 0,X_2\neq 0,\cdots,X_i\neq 0,X_{i+1}=\cdots=X_N=0)\\ &\quad +P(X_1=0,X_2\neq 0,\cdots,X_i\neq 0,X_{i+1}=\cdots=X_N=0)\\ &\quad +\cdots+P(X_1=0,X_2=0,\cdots,X_i\neq 0,X_{i+1}=\cdots=X_N=0)\\ &= P(X_1\neq 0)P(X_2\neq 0)\cdots P(X_i\neq 0)P(X_{i+1}=0)\cdots P(X_N=0)+P(X_1=0)P(X_2\neq 0)\cdots P(X_i\neq 0)P(X_{i+1}=0)\cdots P(X_N=0)+\cdots+P(X_1=0)P(X_2=0)\cdots P(X_i\neq 0)P(X_{i+1}=0)\cdots P(X_N=0)\\ &= P(X_i\neq 0,X_{i+1}=\cdots X_N=0)\\ &= P(X_i\neq 0)P(X_{i+1}=0)\cdots P(X_N=0)\\ &= (1-P)P^{N-i}\end{aligned}$$

设 Ω_b 为在该缓存策略下缓存代理平均需要从服务器获取的补丁数，则：

$$\Omega_b=\sum_{i=1}^{N}iP_i=\sum_{i=1}^{N}i(1-P)P^{N-i}=N-(\frac{P}{1-P})(1-P^N) \tag{9}$$

因此，对于 Nb 大小的补丁窗口，缓存代理需要的最大缓存容量为 Nb，由于从服务器获取的补丁块无重复，$\Omega_b * b$ 即为平均所需的缓存容量。为了计算所需的平均骨干网速率，需要计算两个 RC 之间的平均间隔，若请求在$[Nb,(N+1)b]$到达，则需要在$(N+1)b$ 启动一个新的 RC(其长度为 $T-b$，注意该请求不需要请求补丁)。设 I 为 RC 之间的平均间隔，则：

$$I=Nb+b\sum_{i=0}^{\lfloor\frac{T}{b}-1\rfloor}i(1-P)P^{i-1}=Nb+(1+P+P^2+\cdots P^{\lfloor\frac{T}{b}-1\rfloor})b=(N+1)b+(\frac{p}{1-p})(1-P^{\lfloor\frac{T}{b}-1\rfloor})b$$

因此 IC 算法所需的平均骨干速率为：

$$B_{avg}=\Omega_b b+\frac{(T-b)}{I}r=\frac{[N-P(N+1)+P^{N+1}]b+(T-b)(1-P)}{[N(1-P)+1-P^{\lfloor\frac{T}{b}-1\rfloor+1}]b}r \tag{10}$$

对于在$[ib,(i+1)b]$到达的请求，缓存代理至少需要从服务器获取第$[ib,(i+1)b]$补丁块。因此，服务器服务该请求所使用的平均通道个数等于式(6)，这里则为 $N(1-p)+1$。由于采用了带前缀缓存的批处理补丁，因此客户端的播放延迟为0(这里暂不考虑线路延迟)。显然，由于在补丁窗口内每个请求所需的补丁都尽可能地缓存了，对后续的请求所需的补丁则提高了其传送速度。IC 算法的伪代码描述如算法1所示。

算法1 增量式缓存算法 IC

```
//在同一补丁窗口内的补丁缓存算法
Patch_caching()
{
   while (currBatchPos<=Nb)
   {
      reqCount=Wait(b); //计算间隔 b 内的请求到达个数
      currBatchPos+=b;
      if(reqCount>=1)
      {
         //得到当前批处理需要的补丁长度
         patchLen=getPatchLen(currPatchPos);
         //获取还需从服务器获取的补丁块个数
         fetchLen=getFetchLen(PatchLen);
         //从服务器获取这些补丁块并且缓存起来
         buff=newBuffer(fetchLen);
         fetchAndSave(currBathPos-fetchLen,currBatchPos,buff);
         //更新缓存列表
         up Date cacheList();
```

```
    }
  }
}
```

4.2 缓存及快速释放算法 ICBR

当请求的某个补丁片段不在缓存代理中时，需要由它从服务器获取并且缓存起来。如果此时没有可用的缓存空间，则需要释放某些已经缓存的流数据。在有限的缓存容量的前提下，释放策略直接影响着缓存代理的缓存效率。

假设当前流的当前可用缓存空间为 $C=N'b(N'\in Z^+)$，如果其大于等于补丁窗口大小(即 $N'\geqslant N$))，下一个相邻的窗口内到达的请求可以共享前一个补丁窗口内已经缓存的流数据，因此不需要进行释放。当缓存将整个补丁窗口中的所有数据全部缓存以后，该缓存算法就蜕化为大小为$(N+1)b$的前缀缓存。

如果对特定流媒体对象的缓存代理当前可用空间小于补丁窗口大小(即多个流媒体对象同时使用该缓存空间时)($N'<N$)，在同一补丁窗口中后期到达的请求所需要的补丁在缓存时可能就需要代理缓存的释放操作。假定当前缓存中已经缓存了$[0,N'b]$长度的补丁，当请求在$[N'b,(N'+1)b]$到达时，该请求所需要的是区间$[0,(N'+1)b]$补丁，因此缓存中除了已经缓存的长度为$N'b$数据外，仍需要从源服务器获取$[N'b,(N'+1)b]$的数据。在这种情况下，无论释放$[0,N'b]$中的任何一个长度为b的数据块，都将导致在$[N'b,(N'+1)b]$到达的请求所需的补丁命中率降低。折中的方案是在补丁窗口内的每个空批处理的间隔时将已经缓存的关于该流媒体对象的补丁数据全部释放以保证对其它流媒体对象的请求数据可被缓存。在极端的情况下，当每个批处理间隔都有至少一个请求到达时，该替换算法效率等于 IC 算法。

一般情况下，该释放策略会导致所获取补丁数据的一定程度的重复，增大了骨干网上同一流媒体数据的流量。但在有限的缓存容量的情况下，这种方案获得了一种有效的折中。该缓存释放算法的伪代码描述如算法2。

算法2 缓存及快速释放算法 ICBR

```
//同一补丁窗口内的缓存释放算法
Patch_caching()
{
   while (currBatchPos<=Nb)
   {
      reqCount=Wait(b);//计算间隔 b 内的请求到达个数
      currBatchPos+=b;
      if(reqcount>=1)
      {
          同算法1…
      }
      else
      {
         //获取上一次批处理以后缓存中所有的缓存数据长度
         Buffered=getBuffer();
         if(Buffered!=0)
             free Buffer(0,Buffered);
      }
   }
}
```

4.3 仿真方法及性能评价

我们设计了如下的方法对 IC 算法和 ICBR 算法进行仿真：

第一步，首先我们按照如下算法生成一个泊松请求流：

(1) 对于给定的一个长度为 T 的流媒体对象，假定其请求到达速率为 λ。我们在$[0,1]$上产生$\lfloor \lambda T \rfloor$个随机数，记为$\{R_n,n=1,2,\cdots,\lfloor \lambda T \rfloor\}$；

(2) 对于每个 R_i 计算 $X_i=-(1/\lambda)\ln R_i(i=1,2,\cdots,\lfloor \lambda T \rfloor)$；

(3) 计算请求到达时间 t_i：$t_0=0,t_k=\sum_{i=1}^{k}X_i$，其中 $k=1,2,\cdots,\lfloor \lambda T \rfloor$。

如上所得到的$\{t_n,n=1,2,\cdots,\lfloor \lambda T \rfloor\}$即为本仿真实验中所假定的请求到达序列。

第二步：对于设定的批处理间隔 b 将该请求流划分为多个批处理间隔，如果在间隔$[(i-1)b,ib](i\in Z^+)$至少有一个请求，则在 ib 计算所需获取的间隔长度为 b 的补丁块数并产生一次加入 RC 和取补丁的批处理。对于释放算法，如果在$[(i-1)b,ib]$区间内请求数为0，则释放已经获取并且缓存的补丁数据。

由于批处理补丁调度是在请求到达后的最近一个批处理点将请求加入到 RC 中，因此骨干网络上的数据流量包含 RC 和 PC 两个部分。对于 RC，因其在相邻的补丁窗口之间启动并且传输除了前缀的其它全部流媒体数据。我们认为在 RC 稳定的情况下，骨干网上的其它流量主要取决于补丁块的数量。因此，第三步对于不同的参数 λ,b 和 T 计算缓存代理需要从服务器获取的补丁块数量来仿真上述算法 IC 和算法 ICBR 的性能。

在下面的性能仿真中，我们将补丁窗口设定为批处理补丁的最优窗口 W_{opt}[11]，其计算方法为：

$$W_{opt}=b\left\lfloor \frac{-b+\sqrt{e^{-\lambda b}b^2+2(1-e^{-\lambda b})bT}}{b(1-e^{-\lambda b})}+\frac{1}{2}\right\rfloor$$

图3所示为随着请求速率的变化骨干链路上所需传输的 Patch 数量。不难看出，OBP 随着请求到达速率 λ 的增大所需从服务器获取的补丁数量迅速增加。当 $\lambda\geqslant 5.5$时，补丁窗口中的每个批处理间隔至少有一个请求到达，因此，此时 OBP 算法所需的 Patch 个数收敛于 $\sum_{i=1}^{N}i=N(N+1)/2$，其中 $N=W_{opt}/b$。由实验中我们得出，当 $\lambda=5.5,b=1,T=90$时 $N=12$，此时 OBP 算法需要从源服务器获取的补丁数为78.0。如果对后缀(其长度为 $T-b$)按照 IC 算法进行动态缓存，则随着请求到达速率的增大从服务器获取的补丁数缓慢增加并最终稳定在12.0。当按照 ICBR 算法进行缓存动态缓存和释放时，随着 λ 的增加，其在 $\lambda\leqslant 5.5$时所需从服务器获得的补丁数大于算法 IC，而当 $\lambda>5.5$时，其骨干网带宽的消耗接近于 IC。其中，当 $\lambda=5.5$且 λ 越小时，算法 IC 和 ICBR 表现出越大的差异。这是因为请求到达速率越小，缓存释放频度越大，导致对同一补丁块的传输重复次数越多，因此算法 ICBR 中骨干网上的补丁流量也越大。

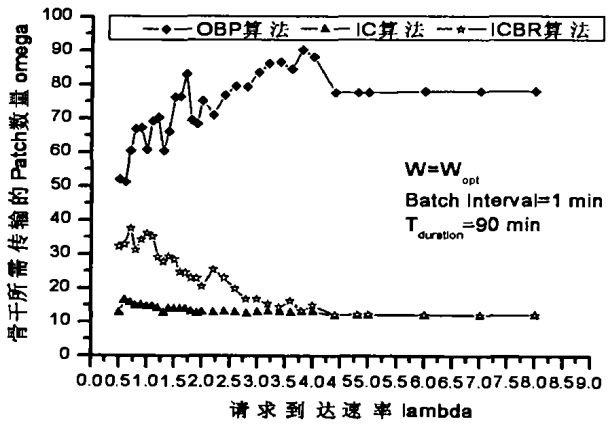
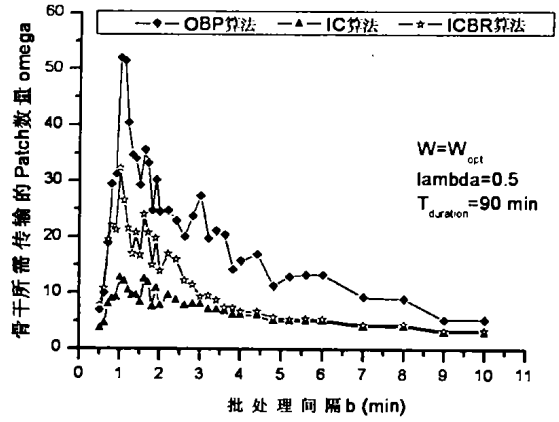
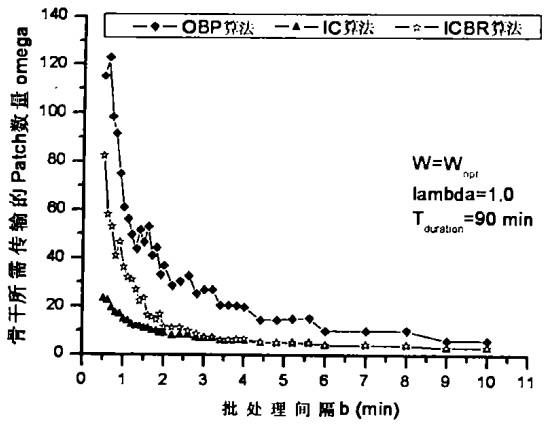


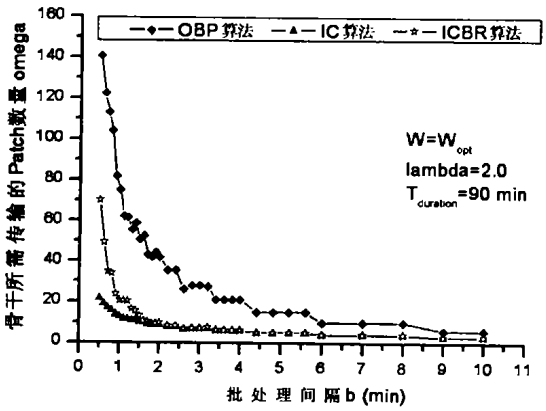
图3 不同的请求到达速率的骨干传输的 Patch 数



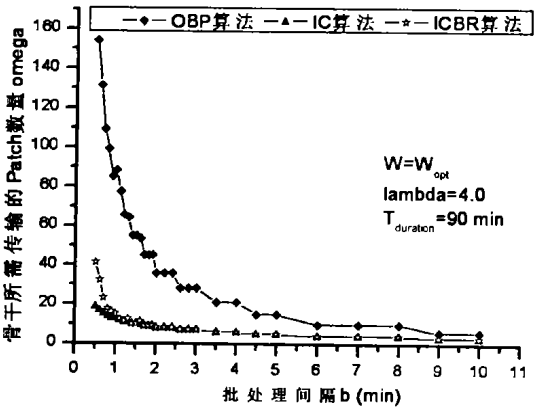
a) $\lambda = 0.5$



b) $\lambda = 1.0$

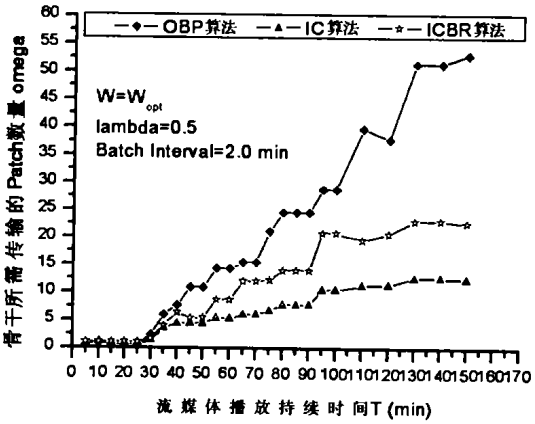


c) $\lambda = 2.0$

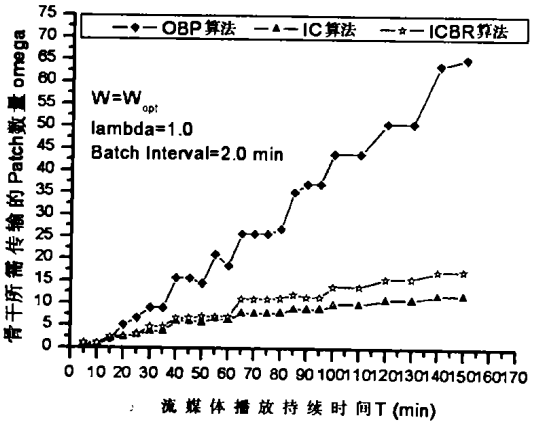


d) $\lambda = 4.0$

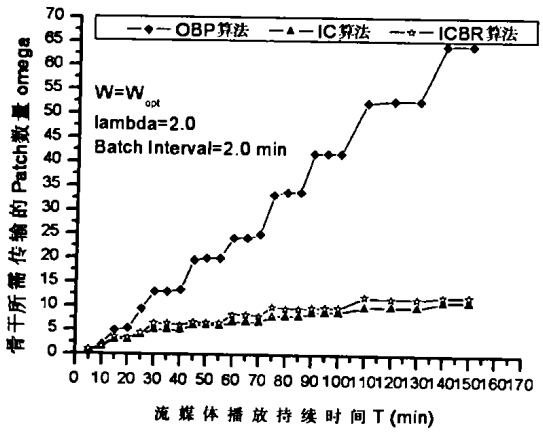
图4　不同批处理间隔的骨干网上传输的 Patch 数



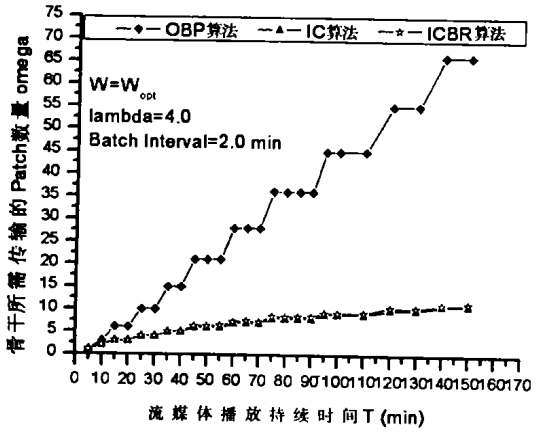
a) $\lambda = 0.5,\quad b = 2\,\text{min}$



b) $\lambda = 1.0,\quad b = 2\,\text{min}$



c) $\lambda = 2.0,\quad b = 2\,\text{min}$



d) $\lambda = 4.0,\quad b = 2\,\text{min}$

图5　不同播放长度流媒体文件的骨干网上传输的 Patch 数

由于请求到达速率 λ 的大小反映了当前被请求流媒体对象的"访问热度"。对于服务器上的不同流媒体文件，我们对 λ 取不同的值来观察算法性能。图4.a)，4.b)，4.c)和4.d)为在不同批处理间隔时，对同一个流媒体对象在 $\lambda=0.5$，$\lambda=1.0$，$\lambda=2.0$，$\lambda=4.0$时的性能。可以看出，在相同的请求到达速率时，随着批处理间隔的增大，缓存代理需要从源服务器获取的补丁数量迅速减少，算法 IC 及 ICBR 的骨干网带宽流量显著低于 OBP。随着请求速率的增加，IC 和 ICBR 的性能趋于接近。

图5.a)，5.b)，5.c)和5.d)是请求不同播放长度的流媒体时骨干所需传输的 Patch 数量。显然，IC 算法和 ICBR 算法均显著优于 OBP 并且随着请求到达速率的增加，IC 算法和 ICBR 算法趋于接近。

结论 如上我们对 IC 算法和 ICBR 算法的性能与 OBP 算法在不同播放长度的流媒体对象，不同请求到达速率和不同批处理间隔时进行了比较。当采用批处理补丁调度策略时，骨干网络上传输的数据为 RC 数据和 PC 数据，对于给定的流媒体对象(T,b 与 λ 给定)，其 RC 流量固定为 T/I。上述结果表明：1)对流媒体对象的后缀采用 IC 及 ICBR 算法进行动态缓存和释放大大降低了骨干传输补丁块的块数，从而有效地降低了骨干带宽的消耗，其性能显著优于 OBP 算法。2)当缓存因容量有限而采用 ICBR 算法对缓存进行管理时，由于骨干网链路上仍需传输部分重复的补丁数据，因此所需传输的补丁块数略大于 IC 算法。ICBR 算法牺牲了部分骨干流量，但是却获得了更高的缓存效率。3)当请求到达速率较大($\lambda>5.5$) 时，会导致每个批处理间隔内均有至少一个请求到达时，此时算法 IC 的性能则等于算法 ICBR 的性能，但都显著优于 OBP 算法。4)(9)式为在每个补丁窗口内平均所需从服务器获得的补丁块数，由于请求到达速率较大时 ($\lambda>5.5$)，算法 IC 的性能接近于算法 ICBR，因此，(9)式也表明了此时系统所需的平均缓存容量。

上述结论说明，对于在存储和传输都占据较大资源量的流媒体对象在 Internet 上的传输，采用带前缀缓存的批处理补丁调度策略并在缓存代理上采用 IC 或 ICBR 算法对补丁数据块进行有效的缓存不仅可以有效降低骨干网带宽和流媒体服务器的负载，同时在保持较低的客户端延迟的前提下，也有效地降低了客户端资源需求(减少了客户端需同时接收流的并发通道个数)和提高了客户端 QoS，从而可更好地支持 VOD 等视频应用。

存储型 A/V 数据是实现 VOD 等应用的数据载体，它是指将视频节目进行编码压缩后存放在媒体服务器上，以按需传输的模式进行的静态的，可按照流式播放的多媒体数据。由于存储型 A/V 数据的本身固有特性和基于包分发的 IP 网络的体系结构及带宽和服务器资源的限制，通过端对端链路的组播来传输存在很大困难，因而一定程度上阻碍了这些应用的发展。而结合应用层组播算法和缓存复制协议的内容分送网络 CDN 进行流媒体数据的分发可以获得骨干网络资源的大量节省，同时也可以有效降低流媒体服务器的负载和提高客户端 QoS，从而更好地支持 VOD 等视频应用。下一步，我们将专注于：(1)研究适合存储型 A/V(Stored A/V Data)数据传输的应用层组播协议及更高效的缓存算法；(2)研究多缓存代理协作体结构和相关算法；(3)结合应用层组播协议和缓存算法研究 CDN 流媒体分发体系结构；(4)设计实现流媒体缓存代理服务器。

致谢 本文工作得到了中国科学院数学所周龙骧教授、中国科学院应用数学研究所安鸿志教授的帮助与指导，其中一些关键结果的推导是与安鸿志教授共同讨论得出的，在此向他们表示衷心的感谢。

参考文献

1 Hua K A, Sheu S. Skyscraper Broadcasting: A New Broadcasting Scheme for Metropolitan VOD systems. In: Proc. of ACM/SIGCOMM, 1997, 27(4): 89～100
2 Viswanathan S, Imielinski T. Metropolitan Area Video-on-Demand Service using Pyramid Broadcasting. Multimedia Systems, 1996, 4(4): 197～208
3 Dan A, Sitaram D, Shahabuddin P. Scheduling Policies for an On-Demand Video Server with Batching. ACM Multimedia, Oct. 1994. 15～23
4 Hua K A, Cai Y, Sheu S. Patching: a multicast technique for true on-demand services. In: Proc. of ACM Multimedia, Sept. 1998. 191～200
5 Cai Y, Hua K A, Vu K. Optimizing patching performance. In: Proc. of ACM/SPIE Multimedia Computing and Networking, Jan. 1999. 204～215
6 Sen S, Rexford J, Towsley D. Proxy prefix caching for multimedia streams. In: Proc. of IEEE INFOCOM'99, 1999
7 Almeida J, Eager D, Vernon M. A hybrid caching strategy for streaming media files. In: Proc. SPIE/ACM Conf. on Multimedia Computing and Networking, Jan. 2001
8 Hofmann M, Ng E, Guo K, Paul, Zhang H. Caching techniques for streaming multimedia over the internet: [Technique Report BL011345-990409-04TM]. Bell Labs. Apr., 1999
9 Sen S, Rexford J, Towsley D F. Proxy Prefix Caching for Multimedia Streams. In: Proc. of IEEE INFOCOM, 1999(3): 1310～1319
10 向哲，钟玉琢，冼伟铨. 一种基于周期合并策略的流调度算法. 软件学报，2001，12(8)：1183～1189
11 White P P, Crowcroft J. Optimised Batch Patching with Classes of Service. ACM SIGCOMM Computer Communication, 2000, 30: 21～28
12 Frossard P, Verscheure O. Batch Patch caching for Streaming Media. IEEE Communications letters, 2002, 6(4)

(上接第18页)

2 Foster I, Kesselman C. The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann Publishers, Inc., San Francisco, CA, 1999
3 Kosar K. Managing and Scheduling Data Placement (DaP) Requests. in Talk - Condor Week, March, 2002
4 Couvares P, Tannenbaum T. Condor Tutorial. In: 1st EuroGlobus Workshop, June 2001
5 Legion: The Grid OS Architecture and User View. in 2nd International Global Grid Forum Meeting, 2001
6 White B, Grimshaw A, Nguyen-Tuong A. Grid-based file access: The Legion I/O Model. In: Proc. of the Ninth IEEE International Symposium on High Performance Distributed Computing, Pittsburgh, PA, August 2000. IEEE Computer Society Press
7 Czajkowski K, et al. A Resource Management Architecture for Metacomputing Systems. In: Proc. IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing, 1998. 62～82
8 Bester J, et al. GASS: A Data Movement and Access Service for Wide Area Computing Systems. Sixth Workshop on I/O in Parallel and Distributed Systems, May, 1999
9 Wang Bin, et al. A Grid-computing-based Solution to Science&Engineering Computing and its Implemenation. ISTM/2003: 5th International Symposium on Test and Measurement, 2003, 6(1): 581～586
10 Ralph B, et al. Components and interfaces of a process management system for parallel programs. Parallel Computing Volume: 2001, 27(11): 1417～1429
11 Grimshaw A S, Wulf W A. The Legion Vision of a Worldwide Virtual Computer, Comm. of the ACM, 1997, 40(1)
12 Thain D, et al. The Kangaroo Approach to Data Movement on the Grid. In: Proc. of the Tenth IEEE Symposium on High Performance Distributed Computing (HPDC10), San Francisco, California, August 2001. 7～9