Journal of Computer Applications
计算机应用,2019,39(7):2001 -2007
ISSN 1001-9081
CODEN JYIIDU
2019-07-10
http://www.joca.cn

文章编号:1001-9081(2019)07-2001-07
DOI:10.11772/j.issn.1001-9081.2019010163

# Gaming@Edge:基于边缘节点的低延迟云游戏系统

林　立,熊金波*,肖如良,林铭炜,陈秀华
(福建师范大学 数学与信息学院,福州 350117)
(*通信作者电子邮箱 jinbo810@163.com)

**摘　要**:云游戏作为云计算的"杀手级"应用正在引领游戏运行方式的变革。然而,云端与终端设备之间较大的网络延迟影响了云游戏的用户体验,因此,提出一种基于边缘计算理念,部署在边缘节点之上的低延迟的云游戏框架——Gaming@Edge。为了降低边缘节点的计算负载以提升其并发能力,Gaming@Edge 实现了一种基于压缩图形流的云游戏运行机制——GSGOD。GSGOD 分离了游戏运行中的逻辑计算和画面渲染,实现了一种边+端的计算融合。此外,GSGOD 还通过数据缓存、指令流水处理以及对象状态延迟更新等机制优化了云游戏的网络数据传输和系统延迟。实验结果表明,Gaming@Edge 相比传统的云游戏系统能够降低平均 74% 的网络延迟,并提高 4.3 倍游戏实例并发能力。

**关键词**:云游戏;云计算;边缘计算;视频流;图形流
**中图分类号**:TP393.09　　**文献标志码**:A

## Gaming@Edge: low latency cloud gaming system based on edge nodes

LIN Li, XIONG Jinbo*, XIAO Ruliang, LIN Mingwei, CHEN Xiuhua
(*College of Mathematics and Informatics, Fujian Normal University, Fuzhou Fujian 350117, China*)

**Abstract:** As a "killer" application in cloud computing, cloud gaming is leading the revolution of way of gaming. However, the high latency between the cloud and end devices hurts user experience. Aiming at the problem, a low latency cloud gaming system deployed on edge nodes, called Gaming@Edge, was proposed based on edge computing concept. To reduce the overhead of edge nodes for improving the concurrency, a cloud gaming running mechanism based on compressed graphics streaming, named GSGOD (Graphics Stream based Game-on-Demand), was implemented in Gaming@Edge system. The logic computing and rendering in the game running were separated and a computing fusion of edge nodes and end devices was built by GSGOD. Moreover, the network data transmission and latency were optimized through the mechanisms such as data caching, instruction pipeline processing and lazy object updating in GSGOD. The experimental results show that Gaming@Edge can reduce average network latency by 74% and increase concurrency of game instances by 4.3 times compared to traditional cloud gaming system.

**Key words:** cloud gaming; cloud computing; edge computing; video streaming; graphics streaming

## 0　引言

随着技术、架构以及商业模式的快速发展和完善,云计算作为一种成熟的计算范型已经得到广泛应用。云计算凭借强大的计算和存储能力提供了分层服务体系,并最终以一切即服务(X as a Service, XaaS)的方式面向终端用户提供便捷、多平台、任意时间和地点访问的服务。云游戏正是近年来出现的被认为是"杀手级"的云计算应用[1]。在云游戏中,当用户请求运行一个游戏时,数据中心启动该游戏实例,执行游戏逻辑,响应用户输入,同时将游戏渲染画面实时压缩以视频流方式传输至用户端。用户接收视频流后可以在任何平台播放游戏画面,如 PC、智能手机、平板电脑、互联网电视等。云游戏的出现改变了以客户端为中心的传统游戏运行模式,它实现了按需游戏的理念(Game on Demand),避免了用户对游戏软硬件的频繁更新,使得用户可以轻松地跨平台同步游戏状态和运行记录[2]。这种新型的云应用目前已吸引了大量游戏玩家并出现了许多成功的商业案例,如 PlayStationNow、LiquidSky、Paperspace 等。

上述的云游戏运行方式,以游戏画面的视频流为基础进行传输,本文称之为"视频流"云游戏。云游戏带来游戏运行方式的革新,但这种基于视频流构建的云游戏存在用户体验的严重不足,即游戏延迟高和带宽需求大。由于云数据中心远离用户终端设备,两者之间的网络往返时间(Round Trip Time, RTT)较大,导致游戏延迟高;同时,由于高清视频编码需要较高的码率,因此云游戏对于用户带宽需求较大,如当前主流的云游戏要求用户至少拥有 3 MB/s 的带宽[3]。此外,当大规模用户请求服务时,数据中心整体的带宽消耗巨大和游戏实例并发能力不足的问题突出。云数据中心基于虚拟化技术为云游戏提供服务[4],而游戏是一种计算和渲染任务都极其繁重的应用,它不但要求 CPU 的逻辑计算能力,同时依赖 GPU 的渲染能力。当前的虚拟化技术对于 GPU 支持不

**收稿日期**:2019-01-22;**修回日期**:2019-03-25;**录用日期**:2019-03-25。　**基金项目**:国家自然科学基金资助项目(61502103, 61872088, 61872086);福建省自然科学基金资助项目(2017J01737);福建省教育厅 A 类资助项目(JAT170140)。

**作者简介**:林立(1982—),男,福建福州人,讲师,博士研究生,CCF 会员,主要研究方向:云计算、边缘计算;　熊金波(1981—),男,湖南益阳人,副教授,博士,CCF 会员,主要研究方向:云计算安全;　肖如良(1966—),男,湖南娄底人,教授,博士,CCF 会员,主要研究方向:云计算、大数据分析。

足[5-6]，导致云游戏数据中心部署成本高昂，游戏并发能力较弱。据报道，曾经著名的云游戏公司 OnLive 就因高昂的系统维护成本导致破产倒闭。

近年来，一种新的计算范型——边缘计算[7-8]的出现，成为解决云游戏高延迟问题极具前景的方式。边缘计算将计算从网络中心（如数据中心）推向网络的边缘（如众多的终端设备）。边缘计算利用边缘节点（如 Cloudlet[9]）的网络邻近性解决了云游戏中的延迟问题，同时这种分布式处理克服了云数据中心集中计算的高带宽消耗。基于这种分析，本文设计和实现了 Gaming@ Edge，一种基于边缘节点的云游戏运行框架。在 Gaming@ Edge 中，云负责用户注册、服务注册、服务发现以及协调各节点的资源调度；边缘节点则作为分布式的计算节点。由于边缘节点计算能力相对有限，框架首先要解决传统视频流云游戏中服务器端负载高、游戏实例并发性低的问题，因此，本文在 Gaming@ Edge 中实现了一种基于图形指令流的云游戏机制——GSGOD（Graphics Stream based Game-on-Demand）。GSGOD 云游戏运行机制继承了云游戏的便利性，游戏运行和维护仍然在服务器端。不同的是在 GSGOD 中服务器拦截游戏的图形渲染指令，以图形流的方式传输至客户端。客户端则完成图形指令的执行，即游戏画面的渲染。GSGOD 分离了游戏逻辑计算和画面渲染任务，既利用边缘节点较强的计算能力，又发挥了客户端的渲染能力，以高效和合理的方式进行端和边缘节点之间的任务分配。这种方式大幅度降低了系统延迟，提升了用户体验，同时成倍地提高了服务器游戏并发性能。此外，为了减少图形流的数据传输量，GSGOD 基于图形渲染指令的统计分析，实现了图形流的压缩；同时优化了图形指令的流水执行和对象状态同步机制，进一步降低了系统延迟。

本文的主要贡献有三点：1）实现了基于边缘节点构建的云游戏框架 Gaming@ Edge；2）实现了基于压缩图形流的云游戏机制 GSGOD；3）系统性对比了图形流和视频流这两种云游戏运行机制的性能。本文所提出的 Gaming@ Edge 验证了基于边缘计算构建云游戏系统的可行性，为后续相关工作提供了经验。

## 1 相关工作

### 1.1 云游戏系统的技术和架构

当前云游戏系统的实现方式可以分为两类：视频流（Video Streaming）方式和图形流（Graphics Streaming）方式[10]。GamingAnywhere（GA）[11]是首个开源视频流云游戏系统，它完整实现了视频流云游戏的各部分组件：客户端的用户输入处理，事件捕捉以及媒体播放；服务器端的音频视频捕捉，基于 H.264/MPEG-4 AVC 的视频压缩以及实时流传输协议。GamingAnywhere 具备良好的可扩展、可移植和可配置等特性，引起学术界和工业界的广泛关注。本文采用 GamingAnywhere 作为视频流云游戏系统解决方案，与 GSGOD 系统进行比较分析。Shi 等[12]在视频流方式基础上提出一种基于 3D 图像变形辅助的实时视频编码方法，该方法使用图形渲染语义提升云游戏的视频编码性能，从而减少了视频数据的传输量和系统延迟。此外，Lee 等[13]以及 Lin 等[14]分别在降低云游戏系统的延迟和提高云游戏服务质量上进行了相关研究。

Games@ Large 项目[15-17]设计了一个通用的云游戏框架，在该框架中用户可以从任何接入设备访问游戏中心并运行游戏。Games@ Large 提出了在服务器端拦截和捕获 3D 图形指令，并以图形流方式传送至客户端进行渲染的方法。该方法与 GSGOD 的原理类似，然而 Games@ Large 基于传统的端云架构，且未对网络传输和延迟进行优化，导致其性能不佳。针对上述缺陷，GSGOD 实现了高效缓存和对象同步机制，从而减小了图形指令和几何数据的传输量，并降低了系统延迟。

### 1.2 边缘计算

边缘计算使得计算发生在网络的边缘，邻近数据产生的地方（即大量的终端设备）。边缘计算的出现源于物联网（Internet of Things，IoT）[18]和 5G 网络的发展[19]，并用于解决实时应用的低延迟需求。边缘计算所提供的低延迟计算依赖于大量部署的边缘节点，边缘节点的形式有多种，包括微云（cloudlet）、网络设备、基站、网络互连车辆[20]等。本文所指的边缘节点即微云，一种可灵活部署的邻近终端设备的“盒中的数据中心”[9]。

边缘计算现在被广泛应用于实时处理应用中。Chen 等[21]设计和实现了 Glimpse，一个持续的、实时的移动对象识别系统。Glimpse 利用计算迁移机制[22]，分离对象识别中对象捕捉和识别算法的执行。移动端负责捕捉动态的视频流以及获取对象的位置，而服务器则执行复杂的对象实时识别算法。Satyanarayanan 团队在实时视频分析应用方面进行了相关研究，开发了 GigaSight[23]，一个多层架构的实时视频流分析系统。GigaSight 系统包含移动设备、边缘节点和云三层架构，采用一种去中心化的计算模式。在这种模式中，移动端捕获的实时视频流数据存储在邻近的边缘节点上，并在边缘节点端进行分析，只有分析的结果和元数据发送到云数据中心，以便后续的大数据分析。云游戏也是一种典型的实时应用，对于网络延迟要求高，而上述应用的构建，为基于边缘计算解决云游戏的高延迟问题提供了参考价值。本文将对这一方案进行讨论和验证。

## 2 系统架构

Gaming@ Edge 的系统框架如图 1 所示。在 Gaming@ Edge 中，云端负责游戏服务的注册和发现；当发现可用的边缘节点后，将用户请求重定向到节点，并由节点提供云游戏的计算服务。本文在边缘节点上实现了一套基于压缩图形流的云游戏系统——GSGOD。GSGOD 不同于传统的视频流云游戏，它实现了一种终端设备迁移游戏逻辑计算至边缘节点，并和边缘节点协同计算完成游戏执行过程的计算范型。该范型既利用云游戏便利、多平台同步的特性，又降低了边缘节点的计算负载。

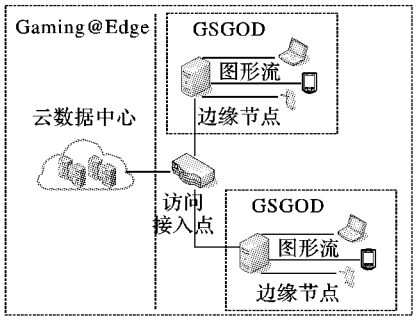


图 1 Gaming@ Edge 系统架构

Fig. 1 Architecture of Gaming@ Edge system

GSGOD 云游戏系统的流程如图 2 所示。整个系统的设计和实现分为两部分：服务器端和客户端。首先客户端请求一个游戏实例，服务器响应请求并启动一个游戏进程。接着客户端和服务器进入一个交互循环，服务器不断接收来自客

户端的输入,客户端则持续执行来自服务器端的图形渲染指令。服务器将接收到的用户输入注入游戏进程,通过 3D 指令代理库拦截 3D 指令,经过缓存处理和数据压缩,形成可由网络传输的指令帧。客户端接收到指令帧之后,进行解压并协同本地缓存还原指令序列,随后这些指令序列被执行后得到游戏画面。

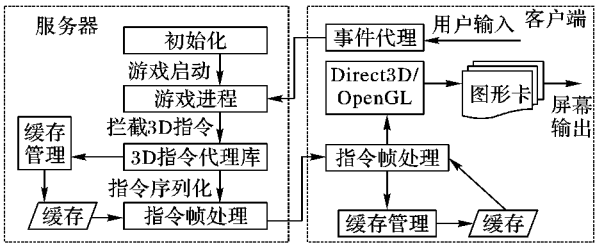


图 2　GSGOD 系统流程

Fig. 2　Workflow of GSGOD system

### 2.1　服务器端实现

服务器端的核心组件是 3D 图形指令代理库。本文实现了一套 3D 指令代理库对系统的图形 API 进行拦截和封装。当前有两种比较常见的 3D 图形 API:Direct3D 和 OpenGL,尽管两者存在许多实现细节上的差异,但基本的渲染机制和图形算法是一致,因此两类图形 API 的代理库实现原理相似。3D 指令代理库继承图形 API 的类和接口,添加了用于重建渲染场景的对象信息,并对 API 的某些行为进行重写,这些修改便于获取场景信息,方便对象管理和对指令帧的处理。GSGOD 采用 hook 机制拦截图形 API 的调用信息,并将系统调用信息传递给代理库,代理库从而接管图形对象的生成和管理。

本文将执行每帧游戏画面渲染的所有指令总和称为指令帧。相邻或间隔相近的指令帧存在许多图形渲染指令和图形几何数据上的相似性。这些冗余信息若未经数据传输优化,将大幅度增加网络带宽,导致网络性能下降。GSGOD 采用高效缓存机制缓存指令和几何数据,数据分别在服务器和客户端缓存,两端保持缓存信息和状态的同步,系统的缓存机制运行逻辑由服务器驱动。关于高效缓存机制的实现细节将在 3.1 节阐述。经缓存机制优化后传输的指令帧在传输之前需要进行压缩,压缩算法必须高效执行以减少频繁处理的延迟,本系统采用 LZO(Lempel-Ziv-Oberhumer)算法进行快速无损压缩。压缩之后的指令帧以网络数据包传输,为了使网络传输有效负荷最大化,GSGOD 以最大传输单元(Maximum Transmission Unit, MTU)为标准划分网络包,同时采用 TCP 协议保证可靠传输。

### 2.2　客户端实现

客户端的主要任务包含:用户输入处理、音频播放和渲染指令执行。GSGOD 使用 SDL(Simple DirectMedia Layer)库(https://www.libsdl.org/)捕捉用户输入事件和播放游戏声音。SDL 是一套跨平台的开发库,支持对音频、键盘、鼠标、控制杆的底层访问,并且可以通过 OpenGL 库和 Direct3D 库访问 GPU。SDL 的跨平台特性使得 GSGOD 具备良好的可扩展性,同时其事件通知机制可以快速响应用户输入,使得用户输入及时传输至服务器从而保证系统的低响应延迟。

游戏画面的渲染是客户端的核心任务。客户端持续接收来自服务器的指令帧数据包,使用 LZO 算法快速解压。解压后的数据包与缓存管理模块交互以正确读取客户端缓存中的指令参数和图形几何数据。

缓存管理模块根据指令帧数据包的相关索引信息快速检索本地缓存,取得缓存数据后整合数据包中已有的数据,形成原始的可执行指令。图形渲染指令的执行依靠具体的图形 API(Direct3D 和 OpenGL),而这两种 3D 图形 API 得到各类 GPU 硬件的广泛支持,并且它们屏蔽了硬件的底层特性,因此 GSGOD 实现方案具备良好的通用性。

## 3　网络传输和响应延迟优化机制

图形渲染指令的正确执行需要包含正确的指令参数、顶点和索引数据以及纹理数据等。这些数据的数据量大,若未经优化进行传输将消耗大量的网络带宽。实验表明,游戏的相邻画面帧之间存在大量的场景相似性,因此图形渲染指令和几何数据大量重叠,这些冗余的信息可以利用缓存机制有效减少。GSGOD 实现了一种高效的服务器和客户端同步缓存机制,该机制能够保证数据的有效缓存以及快速索引和读取。响应延迟的优化包括两个方面:一方面 GSGOD 利用图形指令帧在服务器端和客户端的执行特性,设计了流水线的执行方式,提升了两端的执行效率;另一方面,图形 API 的许多方法调用需要返回图形渲染硬件渲染管线的相关状态,而在 GSGOD 中,图形渲染指令的计算和执行是在服务器和客户端异步执行,服务器如果等待从客户端返回的结果将导致不可容忍的延迟。GSGOD 实现了根据对图形 API 类对象信息追踪和 GPU 硬件状态模拟来快速返回方法调用过程的机制。

### 3.1　高效缓存机制

GSGOD 系统在服务器和客户端实现了一套同步缓存机制,两者在缓存的数据以及命中状态信息上保持一致,但在数据组织和管理上存在差异。服务器负责数据的查找和标识,客户端则根据服务器传输的标识进行数据的快速读取,缓存机制运行的逻辑由服务器驱动。GSGOD 缓存了三类图形相关数据:图形渲染指令参数、顶点数据(vertex)和索引数据(index)。

#### 3.1.1　指令参数缓存

服务器拦截封装的指令帧包含成千上万条的图形渲染指令,这些图形渲染指令多数都包含多个参数,有的指令参数数据量大,可以达到成百上千个字节,于是每一指令帧可能包含数千字节乃至上万字节的参数信息。然而对于一些频繁调用指令,在帧内或相邻帧调用时,它们的参数往往相同。GSGOD 缓存了一些高频调用指令的参数,服务器端使用哈希链表节点保存指令的参数信息,并辅以哈希表管理节点的查找、插入和删除等操作,而客户端则以简单的数组方式存储指令的参数。服务器每拦截一条高频调用指令,便生成其参数的哈希值,并对比哈希链表节点中的哈希码。如果命中某个节点,则服务器生成一个索引值指示客户端缓存数据的具体位置;若没有命中,则向客户端发送原始的参数数据,并更新本地缓存数据。客户端接收到的指令帧包含缓存命中标志位,缓存管理模块检查标志位。若标志位标示缓存命中,客户端按照索引值读取本地缓存的参数值;否则接收网络包中的参数值,并更新本地缓存。客户端的缓存执行逻辑由服务器驱动,不同于服务器哈希表的管理方式,客户端采用数组存储参数值。这样的实现方式降低了客户端的计算负载,同时能够快速访问数据。

#### 3.1.2　顶点和索引数据缓存

顶点(Vertex)和索引(Index)数据是图形渲染指令的重要几何数据,它们存储了顶点坐标、法线向量、切线向量、顶点颜色以及顶点索引值等信息。在复杂场景或精细模型建模的

游戏中,顶点和索引的数据量往往很大,同时相邻和相近帧的数据重复度很高。GSGOD 缓存了一定数量的顶点和索引数据,并且采取增量更新的方法传输更新的数据。服务器计算新的顶点和索引数据与缓存数据的差值,并以地址偏移量标识更新数据的位置,这样不仅缩小了编址长度同时提高了寻址速度。

### 3.2 响应延迟优化

云游戏系统的响应延迟是系统的关键指标,直接影响游戏玩家的体验。在云游戏中,响应延迟定义为从玩家输入控制指令到最终得到游戏画面的时间间隔。GSGOD 系统的延迟组成如图 3 所示,包括在服务器端指令拦截、数据包封装,客户端的指令执行,以及最耗时的网络传输。视频流方式的云游戏系统(典型的如 GamingAnywhere,详见 1.1 节)的响应延迟构成如图 4 所示。这种方式首先在服务器端进行画面渲染,然后通过视频编码生成视频流,随后进行网络传输。

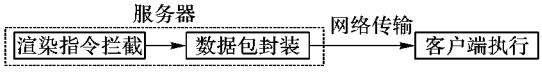


图 3 GSGOD 中响应延迟的主要构成

Fig. 3 Main components of response delay in GSGOD

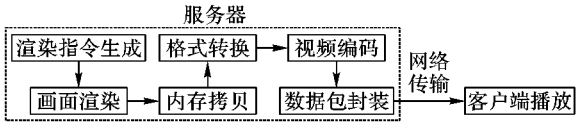


图 4 视频流系统中响应延迟的主要构成

Fig. 4 Main components of response delay in video streaming systems

在 GSGOD 中,为了降低系统延迟,加快云游戏的执行流程,采用了流水线处理机制。图 5 说明了 GSGOD 系统的处理流程,RD(Response Delay)代表系统的响应延迟。服务器的总处理时间用 SP(Server Processing)表示。用户输入经过 ND(Network Delay)网络延迟时间之后到达服务器,服务器将用户输入注入正在运行的游戏之中并执行游戏逻辑,随后开始产生图形渲染指令;经过短暂的启动时间 PB(Pipeline Beginning),图形渲染指令及其相关的图形数据被封装成数据包送往客户端,客户端随即开始执行渲染指令。这里需要注意的是,服务器不会等待一个完整的游戏帧处理结束才开始数据传输,而是当数据量达到一个阈值(比如一个 MTU)就立即开始传输。服务器端生成渲染指令、指令数据网络传输、客户端执行渲染指令三者并发执行,实现了一种处理流水线,并持续一个时间周期 PR(Pipeline Running)。最后,客户端在接收到所有图形数据之后的 PS 时间(Pipeline Stalling),完成游戏帧的渲染,整个客户端的处理时间为 CP(Client Processing)。

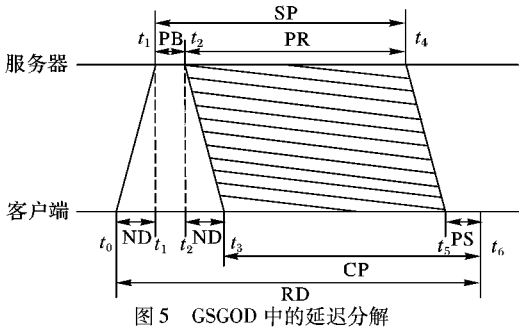


图 5 GSGOD 中的延迟分解

Fig. 5 Breakdown of response delay in GSGOD

视频流方式的云游戏系统的处理流程如图 6 所示。在视频流方式中,服务器花费 SP(Server Processing)时间处理从用户输入注入游戏直至视频流传输至客户端的过程。服务器端的主要处理过程包括执行游戏逻辑从而生成图形渲染指令、游戏画面渲染、内存数据拷贝及格式转换、使用 H.264 协议进行视频编码以及最后的封装数据包进行传输。这些操作在游戏画面帧粒度层次上并行,而 GSGOD 在更细粒度的指令数据包上流水处理,因此 GSGOD 在整体的延迟上较视频流有优势。

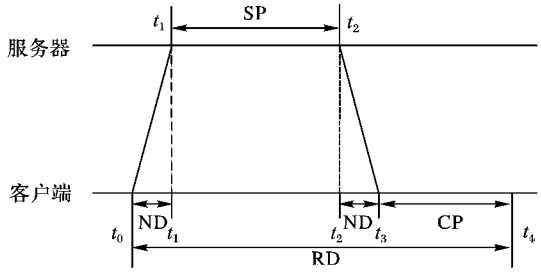


图 6 视频流云游戏中的延迟分解

Fig. 6 Breakdown of response delay in video streaming cloud gaming

### 3.3 对象管理和状态同步

GSGOD 拦截了游戏的图形渲染指令,使指令在服务器和客户端异步执行。服务器并不执行画面的渲染,但其需要掌握当前图形 API 类对象以及 GPU 硬件渲染管线(Pipeline)的状态,而服务器和客户端的异步执行大幅度增加了服务器同步客户端状态的响应时间。GSGOD 维护了服务器和客户端图形 API 类对象列表,在图形渲染指令执行过程中,每一个对象的初始化操作将在服务器和客户端同时生成图形对象,并设置一致的对象 ID 进行对象索引。这种方式保证了类对象的快速调用和状态同步。同时 GSGOD 实现了硬件状态模拟机制:系统初始化时,服务器获取客户端的 GPU 相关参数,这些参数合并服务器自身的 GPU 参数,形成一个标准参数集合;当图形 API 类对象需要获取和设置 GPU 状态时,服务器以自身 GPU 状态模拟客户端硬件执行,并预测对象方法的返回值。该机制确保了 GSGOD 图形渲染指令的快速执行和返回,减少了端到端的延迟。

针对图形渲染指令中一些临界资源(互斥访问资源)的访问操作,GSGOD 实现了一种延迟状态同步机制。这些资源访问操作包含顶点和索引的更新、表面绘制、纹理映射等,这些方法涉及到临界资源的锁(lock)和解锁(unlock)等费时操作。在场景绘制中,通常需要在短时间内频繁地访问临界资源,而在服务器和客户端异步执行时,若每次访问临界资源都执行端到端的状态同步,将导致大量的时间开销。在实际运行中,上述的资源访问操作在多数情况下并未立即生效,因此 GSGOD 实现了一种端到端的状态延迟同步机制,即直到资源状态真正改变时才执行同步。延迟同步机制合并了资源访问操作,减小了频繁资源更新的时间开销。

## 4 系统评估

Gaming@Edge 中游戏服务注册,边缘节点的注册和发现以及资源管理采用 Python 语言实现,而 GSGOD 目前实现了 Direct3D 的版本,基于 C++ 语言实现,代码超过 20 000 行。本章从三个方面对 GSGOD 的性能进行评估:游戏实例的服务器并发测试、系统响应延迟以及缓存机制性能。实验搭建了一台具有高性能 GPU 的服务器作为边缘节点,服务器使用 VMware Workstation 技术提供虚拟桌面连接,宿主机和虚拟主机均采用 Windows 7 操作系统,每个客户端连接一个虚拟主机,具体的实验环境配置参见表 1,实验平台的部署如图 7 所

示。

表 1　实验环境配置

Tab. 1　Configuration of experimental environment

| 实验平台 | CPU | 内存/GB | GPU |
|---|---|---|---|
| 服务器 | Intel Xeon E5620, 2.4 GHz,4 核心,8 线程 | 12 | 华硕 Radeon HD 6990 |
| 客户端 | Intel Core i7,2.4 GHz | 4 | Intel HD Graphics 4000 |

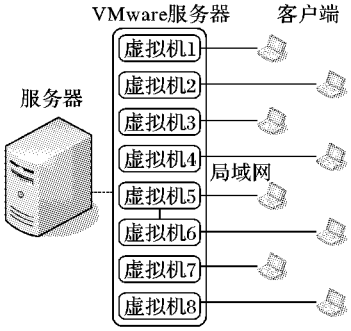


图 7　实验平台部署

Fig. 7　Deployment of experimental platform

本实验选取了三类不同的热门 3D 游戏进行系统测试,分别是休闲类游戏 SprillRitchie、策略类游戏 CastleStorm 和动作类游戏 Trine。经实验测试,游戏 CastleStorm 和 Trine 的逻辑计算和渲染计算复杂度均高于SprillRitchie。为了对比视频流云游戏系统的性能,本实验采用 GamingAnywhere (GA)[11]作为视频流云游戏机制的参照系统。如 1.1 节所述,GA 是学术界知名的开源视频流云游戏系统,由台湾几所大学联合开发。GA 系统实现了视频流云游戏机制的完整功能,即客户端的输入处理和画面输出,服务器的游戏执行、画面渲染以及视频流的压缩和传输;同时,它还优化了系统的扩展性、移植性和配置性。

### 4.1　游戏实例并发测试

本节测试服务器游戏实例的并发能力,即测试当客户端连接数目不断增加时(服务器并发的游戏实例逐渐增加),服务器的帧速率(Frames Per Second, FPS)、CPU 利用率和 GPU 利用率的变化情况。帧速率表明了图形处理器每秒能够渲染刷新的画面帧数目,它是一种综合体现图形处理设备渲染能力的指标。帧速率高于每秒 10 ~ 12 帧时,画面被认为是连贯的,一般认为 FPS 大于 25 帧时,游戏画面是流畅的。在并发测试中,每个客户端连接一个虚拟机(Virtual Machine, VM),每个虚拟机运行一个游戏实例,一个虚拟机分配 1 GB 内存。图 8 分别显示了 3 个游戏在虚拟机个数从 1 到 8 情况下服务器的性能表现,其中 FPS 为计算所有虚拟机帧速率的平均值。实验结果表明 GSGOD 系统在三个指标上均优于 GA,即服务器整体的负载小,游戏运行综合性能好。这主要因为 GSGOD 实现的图形流云游戏机制采取游戏逻辑和图形渲染分离的机制,并由客户端执行图形渲染,从而降低了服务器的负载,大幅度提升了服务器的整体性能。

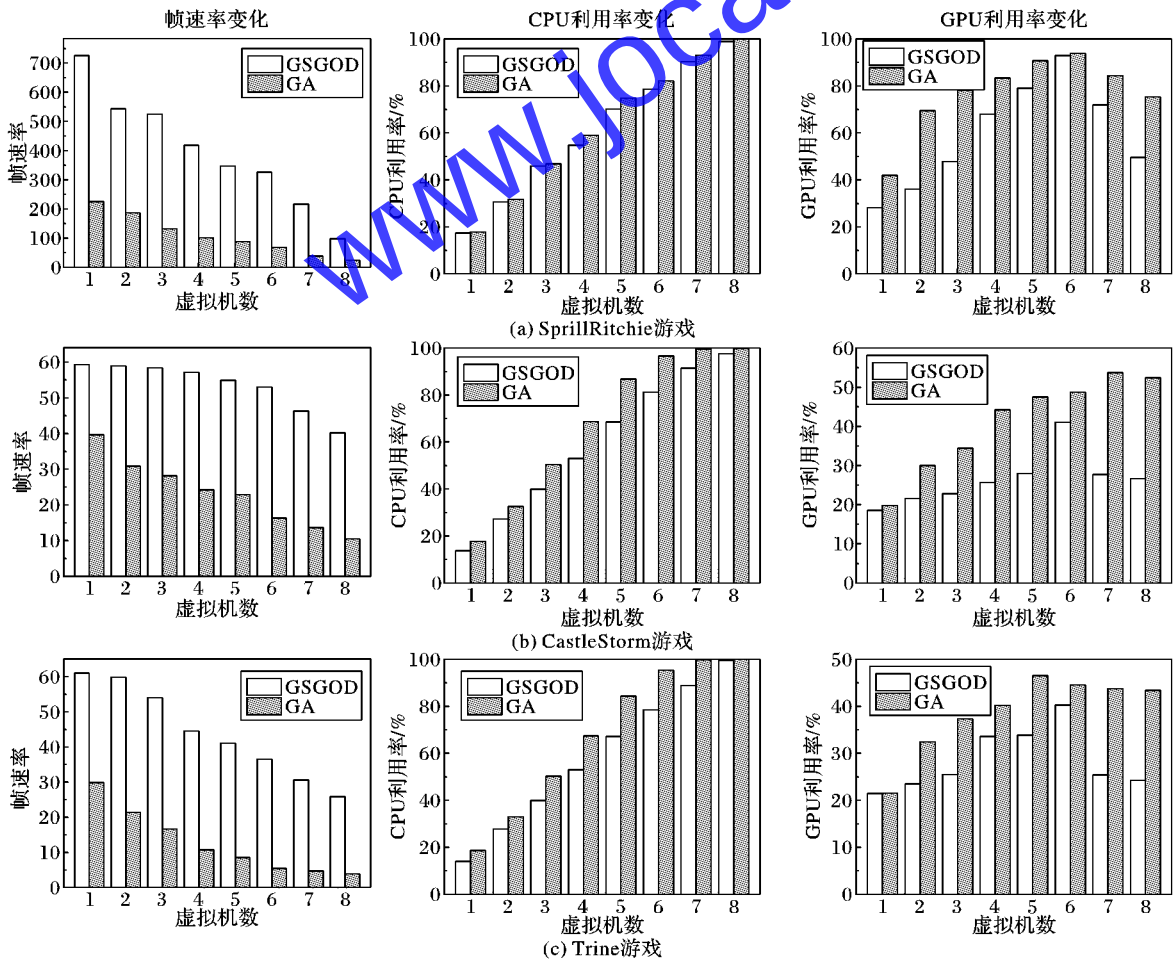


图 8　3 个游戏在不同虚拟机个数下服务器的性能表现

Fig. 8　Performance of three games under different number of virtual machines

#### 4.1.1 游戏实例并发性能

游戏实例并发性能反映了服务器运行的综合性能，即在相同硬件配置下，服务器能够运行的最大有效游戏实例数目。这里“有效”游戏实例指游戏运行时 FPS 能够大于某个阈值（以 25FPS 为基准），达到运行流畅的标准。表 2 对比了 GSGOD 和 GA 系统在 CastleStorm 和 Trine 中（SprillRitchie 逻辑较为简单，当服务器达到最大承载虚拟机数时，其 FPS 仍高于游戏流畅标准，因此不作对比。）游戏实例并发性能，可以看出 GSGOD 分别取得 1.67 倍和 7 倍的性能提升。

**表 2 两个系统游戏实例并发性能比较**

Tab. 2 Comparison of game instance concurrency performance of two systems

| 游戏 | GSGOD 并发数 | GA 并发数 | 并发性能提升倍数 |
|---|---|---|---|
| CastleStorm | 8 | 3 | 1.67 |
| Trine | 8 | 1 | 7.00 |

#### 4.1.2 CPU 和 GPU 利用率

GSGOD 不仅获得了良好的游戏 FPS 性能，同时其服务器的 CPU 利用率也相对较低。GSGOD 的 3D 指令代理库在拦截渲染指令时，增加了额外的计算开销，但是这些开销低于 GA 在游戏画面视频编码时产生的计算开销，因此 GSGOD 系统服务器 CPU 计算负载更低。随着游戏复杂度的增加，这种趋势更加明显。

实验结果中三个游戏的 GPU 利用率呈现出不同的变化趋势，但总体看来 GSGOD 仍然显著低于 GA 系统。GPU 利用率跟游戏自身特性以及游戏的帧速率密切相关。通常情况下，当游戏场景越复杂、帧速率越高时，GPU 渲染的负载越高。SprillRitchie 是休闲类游戏，画面复杂度低，GPU 能够以较少的处理时间高速渲染游戏场景，因此表现出很高的帧速率和较高的 GPU 利用率，而 CastleStorm 和 Trine 两款游戏由于场景复杂，对象模型数据量大以及众多的纹理和光影特效等，使得 GPU 渲染游戏的负载增长较为平稳。三个游戏的 GPU 利用率变化先是随着游戏实例数的增加而升高，达到一定阈值之后，实验硬件设备达到较大的负载，随后游戏的整体帧速率下降明显，GPU 利用率也随之下降。在 GSGOD 中，虽然服务器端不执行图形的渲染，但是一些图形操作，如图形指令生成、模型加载以及纹理的处理等仍然需要大量 GPU 的参与，因此当游戏实例数增加时，GPU 的负载随之增加。由于 GSGOD 系统迁移了大部分的渲染任务，因此其 GPU 利用率较 GA 有明显的下降；同时视频流方案中，游戏画面的实时压缩使得渲染数据在 GPU 专属内存和系统内存中频繁拷贝，导致了系统处理时间增加，而多个虚拟机共享 GPU 也加剧了其性能的下降。

### 4.2 系统响应延迟

云游戏系统作为一个实时系统，对系统的响应延迟有很高的要求，不同类型的游戏对于端到端延迟有明确限定[24]。云游戏的响应延迟定义为：从客户端接收一个用户输入，直至用户最终在客户端看到由该输入触发的游戏画面帧为止的时间间隔[25]。GSGOD 系统的响应延迟包含三部分：网络传输延迟，通常情况下可以认为是网络的往返时延 RTT；服务器处理用户请求的时间，即服务器接收用户输入并注入到游戏中，随后执行逻辑处理，完成某一特定帧渲染指令的封装，并传输该帧数据包的总时间开销；客户端完成帧渲染的时间，即客户端接收到来自服务器的数据包并调用图形 API 完成帧渲染的时间。GA 系统的响应延迟划分与 GSGOD 类似，但是其服务器处理请求需要完成画面渲染和视频压缩，而客户端则是播放视频。关于两个系统的延迟构成细节可以参阅 3.2 节。

测试响应延迟是一项复杂的工作，主要是由于系统很难捕获用户输入所指向的特定帧。文献[25]通过测定特殊画面（如游戏菜单选择界面）的像素值变化来判断用户输入所产生的渲染画面，该方式具有一定的特殊性，无法测试游戏运行过程中不同场景的延迟。本实验的方法是通过向服务器发送特殊的键盘输入（如 F8），服务器接收到该输入指令后，在当前帧的下一帧执行输入请求，完成指令封装或者帧渲染处理之后立即将数据返回给客户端，其中的时间间隔即为该输入产生的延迟。本实验分别对三个游戏进行 300 次延迟采样，图 9 对比了两个系统的平均响应延迟（300 次延迟的平均值）。结果表明 GSGOD 的平均响应延迟远低于 GA，主要是由于 GSGOD 端到端更快的流水线处理（如 3.2 节所述）。在 GSGOD 系统中，服务器渲染指令的拦截，数据包的封装和发送可以快速启动而不必等待完整一帧指令全部产生，即一旦指令产生，就可以开始数据封装和网络传输。该方式加快了服务器和客户端的渲染同步，降低了系统响应延迟。

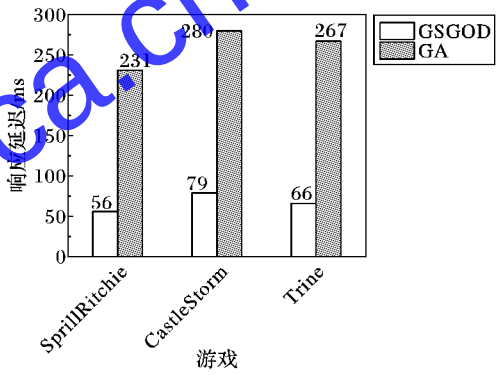


图 9 两个系统的平均响应延迟对比

Fig. 9 Average response delay comparison of two systems

### 4.3 缓存机制性能

缓存机制在 GSGOD 系统中发挥重要的作用，本节评估了缓存机制的性能。实验针对 GSGOD 在游戏运行过程中 Direct3D 库高频调用方法的一个子集：{SetIndices, SetStreamSource, SetTexture, DrawIndexedPrimitives} 的缓存性能进行了评估。实验中，分别运行三个游戏 5 min，记录其总渲染帧数、高频子集的总调用次数和总命中次数，实验结果如表 3 所示。结果表明，高频调用方法集在游戏渲染中调用频繁，并且指令参数缓存机制具备很高的命中率，将节省大量的网络传输带宽。

**表 3 GSGOD 缓存机制性能**

Tab. 3 Performance of GSGOD cache mechanism

| 游戏 | 总帧数 | 调用次数 | 命中数 | 命中率/% |
|---|---|---|---|---|
| SprillRitchie | 19 100 | 2 537 566 | 2 537 079 | 99.98 |
| CastleStorm | 7 758 | 5 780 983 | 5 778 937 | 99.96 |
| Trine | 5 771 | 6 300 060 | 6 260 291 | 99.36 |

## 5 结语

云游戏是云计算的“杀手级”应用，它带来了游戏运行方

式的革命性变革,但是受限于客户端与云数据中心较大的响应延迟,云游戏体验不佳。本文提出一种基于边缘节点部署的云游戏运行框架——Gaming@ Edge,该框架能够有效降低系统的延迟;同时,为了降低边缘节点的计算负载,Gaming@ Edge 设计和实现了一种基于压缩图形流的云游戏运行机制——GSGOD。GSGOD 通过拦截游戏渲染的 3D 图形指令,并迁移至客户端进行本地渲染的方式,实现了一种客户端和边缘节点的计算融合。这种方式既继承了云游戏诸多优点,又有效降低了边缘节点的计算负载。此外,GSGOD 通过高效的缓存机制优化了网络数据量的传输,同时基于指令流水线处理、对硬件状态模拟的异步执行方式,以及延迟状态同步等方法,降低了系统端到端的延迟。最后,本文从系统的并发性、游戏的响应延迟以及缓存性能等三个方面综合评估了系统性能。实验结果表明 GSGOD 显著提升了服务器游戏实例并发性能,同时获得较小的系统响应延迟。

**参考文献**(References)


[1] ROSS P E. Cloud computing's killer app: gaming [J]. IEEE Spectrum, 2009, 46(3): 14.

[2] CAI W, SHEA R, HUANG C Y, et al. The future of cloud gaming point of view] [J]. Proceedings of the IEEE, 2016, 104(4): 687 -691.

[3] USMAN M, IQBAL A, KIRAN M. A bandwidth friendly architecture for cloud gaming [C]// Proceedings of the 2017 IEEE International Conference on Information Networking. Piscataway, NJ: IEEE, 2017: 179 -184.

[4] SHEA R, LIU J C, NGAI E C H, et al. Cloud gaming: architecture and performance [J]. IEEE Network, 2013, 27(4): 16 -21.

[5] DOWTY M, SUGERMAN J. GPU virtualization on VMware's hosted I/O architecture [J]. ACM SIGOPS Operating Systems Review, 2009, 43(3): 73 -82.

[6] ZHANG W, LIAO X F, LI P, et al. ShareRender: bypassing GPU virtualization to enable fine-grained resource sharing for cloud gaming [C]// Proceedings of the 25th ACM International Conference on Multimedia. New York: ACM, 2017: 324 -332.

[7] 施巍松,孙辉,曹杰,等. 边缘计算:万物互联时代新型计算模型[J]. 计算机研究与发展,2017,54(5):907 -924. (SHI W S, SUN H, CAO J, et al. Edge computing—an emerging computing model for the Internet of everything era [J]. Journal of Computer Research and Development, 2017, 54(5): 907 -924.)

[8] SHI W S, CAO J, ZHANG Q, et al. Edge computing: vision and challenges [J]. IEEE Internet of Things Journal, 2016, 3(5): 637 -646.

[9] SATYANARAYANAN M, BAHL V, CACERES R, et al. The case for VM-based cloudlets in mobile computing [J]. IEEE Pervasive Computing, 2009, 8(4): 14 -23.

[10] LIAO X F, LIN L, TAN G, et al. LiveRender: a cloud gaming system based on compressed graphics streaming [J]. IEEE/ACM Transactions on Networking, 2016, 24(4): 2128 -2139.

[11] HUANG C Y, HSU C H, CHANG Y C, et al. GamingAnywhere: an open cloud gaming system [C]// Proceedings of the 4th ACM Multimedia Systems Conference. New York: ACM, 2013: 36 -47.

[12] SHI S, HSU C H, NAHRSTEDT K, et al. Using graphics rendering contexts to enhance the real-time video coding for mobile cloud gaming [C]// Proceedings of the 19th ACM International Conference on Multimedia. New York: ACM, 2011: 103 -112.

[13] LEE K, CHU D, CUERVO E, et al. Outatime: using speculation to enable low-latency continuous interaction for mobile cloud gaming [C]// Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services. New York: ACM, 2015: 151 -165.

[14] LIN Y, SHEN H. CloudFog: leveraging fog to extend cloud gaming for thin-client MMOG with high quality of service [J]. IEEE Transactions on Parallel and Distributed Systems, 2017, 28(2): 431 -445.

[15] NAVE I, DAVID H, SHANI A, et al. Games@ Large graphics streaming architecture [C]// Proceedings of the 2008 IEEE International Symposium on Consumer Electronics. Piscataway, NJ: IEEE, 2008: 1 -4.

[16] EISERT P, FECHTELER P. Low delay streaming of computer graphics [C]// Proceedings of the 15th IEEE International Conference on Image Processing. Piscataway, NJ: IEEE, 2008: 2704 -2707.

[17] JURGELIONIS A, FECHTELER P, EISERT P, et al. Platform for distributed 3D gaming [J]. International Journal of Computer Games Technology, 2009, 2009(1): 1 -15.

[18] GUBBI J, BUYYA R, MARUSIC S, et al. Internet of Things (IoT): a vision, architectural elements, and future directions [J]. Future Generation Computer Systems, 2013, 29(7): 1645 -1660.

[19] HU Y C, PATEL M, SABELLA D, et al. Mobile edge computing—a key technology towards 5G [J]. ETSI White Paper, 2015, 11(11): 1 -16.

[20] 刘小洋,伍民友. 车联网:物联网在城市交通网络中的应用[J]. 计算机应用,2012,32(4):900 -904. (LIU X Y, WU M Y. Vehicular CPS: an application of IoT in vehicular networks [J]. Journal of Computer Applications, 2012, 32(4): 900 -904.)

[21] CHEN T Y, RAVINDRANATH L, DENG S, et al. Glimpse: continuous, real-time object recognition on mobile devices [C]// Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems. New York: ACM, 2015: 155 -168.

[22] DINH H T, LEE C, NIYATO D, et al. A survey of mobile cloud computing: architecture, applications, and approaches [J]. Wireless Communications and Mobile Computing, 2013, 13(18): 1587 -1611.

[23] SATYANARAYANAN M, SIMOENS P, XAIO Y, et al. Edge analytics in the Internet of things [J]. IEEE Pervasive Computing, 2015, 14(2): 24 -31.

[24] CLAYPOOL M, CLAYPOOL K. Latency and player actions in online games [J]. Communications of the ACM, 2006, 49(11): 40 -45.

[25] CHEN K T, CHANG Y C, TSENG P H, et al. Measuring the latency of cloud gaming systems [C]// Proceedings of the 19th ACM International Conference on Multimedia. New York: ACM, 2011: 1269 -1272.


This work is partially supported by the National Natural Science Foundation of China (61502103, 61872088, 61872086), the Natural Science Foundation of Fujian Province (2017J01737), the Education Department Foundation of Fujian Province (JAT170140).

**LIN Li**, born in 1982, Ph. D. candidate, lecturer. His research interests include cloud computing, edge computing.

**XIONG Jinbo**, born in 1981, Ph. D., associate professor. His research interests include cloud computing security.

**XIAO Ruliang**, born in 1966, Ph. D., professor. His research interests include cloud computing, big data analysis.