A Distributed Render Farm System for Animation Production

Jiali Yao, Zhigeng Pan[*], Hongxin Zhang

State Key Lab of CAD&CG, Zhejiang University, Hangzhou, 310058, China
{yaojiali, zgpan, zhx}@cad.zju.edu.cn

Abstract. Render farm is widely used in movie industry to solve the long rendering time problem. By parallel computing rendering jobs, render farm can speedup the rendering process in a scalable way. In this paper, we present an efficient design of render farm system name DRFarm in distributed environment. The most important feature of the system is the capacity aware task scheduling strategy. We first introduce the hierarchy tasks subdivision method which ensures flexible merging and dividing tasks. By carefully grouping tasks and dynamically assign them in different modes, the overall parallel rendering time can be reduced by exploiting coherence comparing to conventional methods. Furthermore, to adopt various rendering jobs from different locations, we design a general rendering service interface with unified job definition.

Keywords: parallel rendering, distributed rendering, render farm

1. Introduction

In movie industry, rendering a full-length animation film cost numerous CPU time. Render farm is built to reduce the rendering time by parallel computing individual frame in distributed environment.

Typical render farm is constructed for animation film rendering. The first full-length animation film, Toy Story, used 117 Sun workstations and the Pixar RenderMan system. The film, Shrek 3, is rendered by more than 4000 HP workstations, where every second requires 3000 hours CPU time. Render farm is also widely used in architecture design, advertising, and the visual effects industry.

Different from other massive parallel systems, render farm emphasizes on cost-effective ratio, and usually adopts contemporary commodity workstations. Although multiprocessor platform performs excellent for parallel rendering, especially interactive ray tracing applications [1], render farm focus on high quality off-line batch rendering.

Our render farm system which named DRFarm is built on commodity multi-core PCs connected by commodity Ethernet infrastructure. In our implementation, we focus on scalability and rendering client utilization by adopting hybrid task schedule method and flexible task grouping strategy.

[*] Corresponding author

2. Related Works

The “render farm” concept has long been applied in movie industry to achieve high quality images by parallel computing. The “Kilauea” renderer parallel computes global illumination algorithms on cost-efficient PCs running Linux, but not considered for real-time purpose. Recently, interactive ray tracing system has been implemented from SGI workstations to even commodity PC clusters [2]. These systems use highly custom optimized renderers. Specially optimized renderers help exploit hardware feature, such as SIMD units [3], but loose certain flexibility.

Commercial render farm management systems include Qube from PipelineFX, Enfuzion from Axceleon, Muster from Virtual Vertex, and Deadline from Frantic Film. There are also open source render farm systems such as drqueue. These systems generally support as much renderers as possible for different users, and focus on service quality. However, such system usually split tasks into single frame, further optimization have to be manually configured if possible.

Online render services are provided by remote render farms, such as RenderRocket, which helps make global utilize of computing resource. Grid [4] or volunteer computing [5] are also suitable for remote rendering. Implementing such remote parallel systems usually requires design network interface and handle data access in complex environment [6].

In our work, we offered novel strategy to handle task schedule. Hierarchy task subdivision definition is introduced to ensure both fine and coarse granularity. With flexible task merging and subdividing method, tasks can be grouped together aware of client’s computing capacity to exploit temporal coherence between frames.

3. System Design

3.1 System Overview

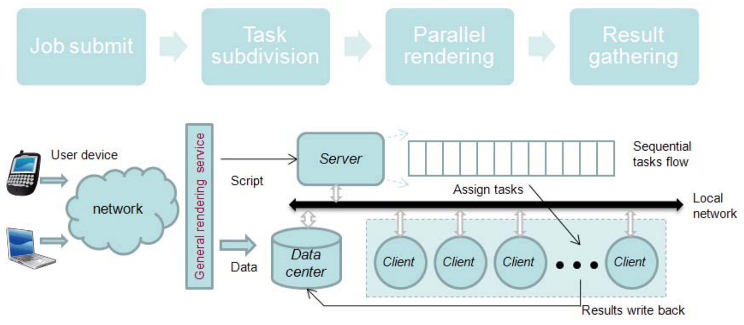

Fig. 1. The architecture of the render farm system

Figure 1 is the architecture of the render farm system (DRFarm). In the system, the server manages all the storage and rendering resources, and provides services to remote or local users by general render service (GRS) interface. In the render farm, hardware is connected by local network infrastructures.

We implement network architecture in C/S mode, and TCP/IP is employed to handle communication. Most other render farm systems use local network file sharing strategy for scene data access, which is based on SMB in windows operation system or Samba in Linux OS. We implement both to avoid network bottleneck.

3.2 Hierarchy Task Subdivision

Renderers used in render farm system are generally based on ray-tracing algorithms which are famous for embarrassing parallelism. When each client has necessary data resources, pixels can be computed independently. For the average user, some popular commercial animation packages employ coarse-grain parallelism to allow rendering of individual frames of an animation across a network of machines. Other renderers such as POV-Ray [8], parallelization scheme works on single images only.

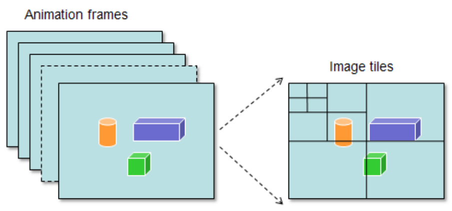

Fig. 2. Temporal and spatial parallelism in animation

Among all the parallel methods, we focus on temporal parallelism between frame sequences and spatial parallelism inside single frame. Since animation frame has inherent corresponding to each other, we employ a hybrid method to dynamically subdivide single frame into finer granular tasks.

Modern hardware architecture benefits from hierarchical arrangement of different levels parallelism. For example, the Cell cluster has three levels hardware parallelisms to exploit [7]. Furthermore, tasks can be expanded and merged in a flexible manner, which is a requirement of capacity aware tasks assign algorithm.

3.3 Dynamic Load Balance

Granularity of tasks greatly affects load balance performance. Conventional method in commercial render farm management system is setting up a task pool and dividing the tasks into a modest but fixed granularity such as one frame or two. With dynamic task subdivision method, we employ capacity aware strategy which will be introduced in experiment section.

We design load balance algorithms for both job completion oriented and client utilization ratio oriented requirements, named active mode and passive mode separately.

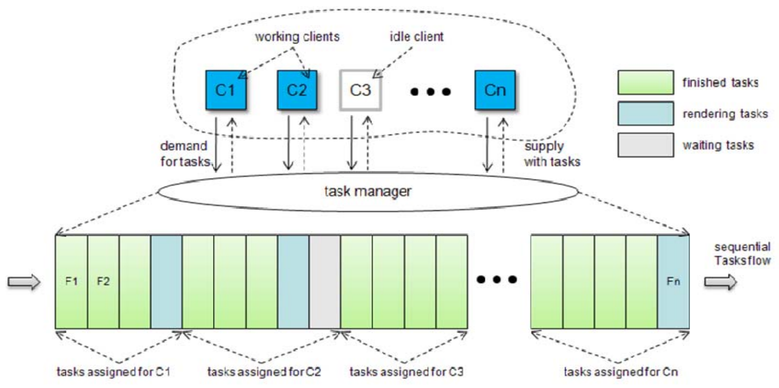

Fig. 3. Dynamic task schedule in active mode

In active mode, the server groups and distributes tasks to rendering clients by their rendering capacity. If certain client finished all the tasks assigned, it will search the queue to find unfinished tasks assigned to other clients. In passive mode, all tasks will be pushed into the task queue. Rendering client fetches tasks each time when its current state is idle. The passive mode doesn't guarantee job completion in coarse granularity.

3.4 Unified Job Submission

Supporting various kinds of renderers is the key requirement in the render farm system. We define an abstract level between actual renderer and user, which named general rendering service interface (GRS). After analyzing, we define jobs in quintuple form: Job = {Type, Job decomposition, Renderer, Data access, Result composition}. The five elements are the minimal elements of submitted job script.

Figure 1 illustrates the functionality of GRS. Users located in different area can submit various types of Jobs via GRS interface. A script of the current job will be generated including all necessary items for data access and task subdivision. The script is coded in XML format, thus can be easily decode and transferred into other render farm platform script

4. Experiment

In practice, the computing capacity of rendering clients might be different. Based on the fact that frequent memory operation and communication between rendering clients will cause the system inefficient, we present a capacity aware grouping strategy. Instead of using third party tools to measure performance, we firstly run fine grain tasks to collect overall rendering information of each rendering client. With the heuristic statistics, new tasks are grouped together as coarse grain ones to exploit coherence and reduce loading time.

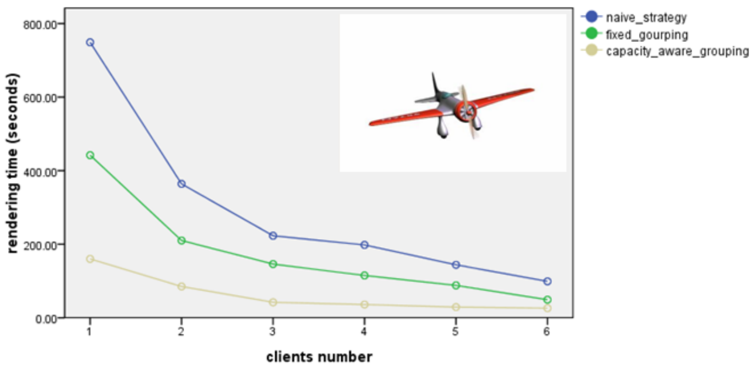

Fig. 4. Rendering time of a 48 frame simple Maya animation.

To illustrate how the flexible tasks subdivision and grouping algorithm work, we task a 48 frames animation (plane fly animation as shown above) for example, which is rendered by Maya renderer. The experiment is running on 6 different workstations.

In this example, compared with scene loading time, rendering time of one frame is very short. The naïve strategy split tasks into single frames thus caused great loading time consumption. The green line is the fixed task group in size of 2. As shown above, capacity aware grouping strategy has flexible task size according to rendering clients' capacity and outperforms other two conventional strategies.

After repeating the naïve while the finest subdivision algorithm several times, we can find that client with more computing power tends to compute more frames. Figure 5 is the average frames rendered at each client.

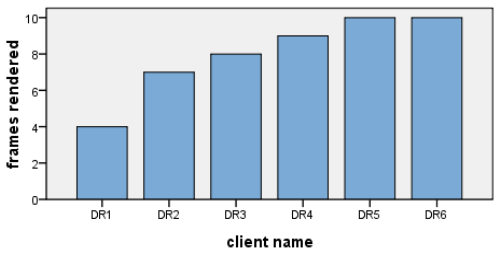

Fig. 5. Client capacity represented by frames computed.

Client capacity can be evaluated by clients' rendering history statistics. With client rendering capacity in mind, the system therefore subdivides frames into finest granularity and then merges them by capacity value. As shown in Figure 4, capacity aware strategy can achieve very excellent speed-up.

5. Conclusion and future work

In this paper we present the fundamental concept for building a render farm system. We have shown that commodity PC cluster also performances well as a distributed parallel rendering environment. Although increasing the client number in straight forward way will not effectively promote speedup limited by network I/O bandwidth, acceptable scalability can still be achieved using methods like grouping and coherence exploiting.

Currently, render farm is still limited in local area network. However grid and volunteer computing systems can help utilizing remote resources. Future work can focus on idle PCs based remote rendering.

References

1. Bigler, J., Stephens, A., Parker, S.G.: Design for parallel interactive ray tracing systems. In: Proceedings of the IEEE Symposium on Interactive Ray Tracing, pp. 187--195 (2006)
2. Wald, I., Benthin, C., Dietrich, A., Slusallek, P.: Interactive Distributed Ray Tracing on Commodity PC Clusters - State of the Art and Practical Applications. Lecture Notes on Computer Science 2790, pp. 499--508 (2003)
3. Benthin, C., Wald, I., Scherbaum, M., Friedrich, H.: Ray tracing on the CELL processor. In: Proceedings of the IEEE Symposium on Interactive Ray Tracing, pp. 15--23 (2006)
4. Foster, I., Kesselman, C.: The Grid 2: Blueprint for a new computing infrastructure. Morgan Kaufmann, San Fransisco (2004)
5. Anderson, D.: BOINC: A System for Public-Resource Computing and Storage grid. In: Fifth IEEE/ACM International Workshop on Grid Computing, pp. 4--10. IEEE Computer Society, Washington, DC (2004)
6. Pan, Z., Shi, J., Zhang, M.: Distributed graphics support for virtual environments. In: Computers & Graphics vol. 20, no. 2, pp.191--197 (1996)
7. Komatsu, K., Takizawa, H., Kobayashi, H.: Hierarchical Parallel Processing of Ray Tracing on a Cell Cluster. In: 1st International Workshop on Super Visualization, CD-ROM, (2008)
8. POV-Ray documentation, http://www.povray.org/documentation/view/3.6.0/7/