Eventful Cloud Service

Ibraheem Alhashim
School of Computing Science
Simon Fraser University
Surrey, BC, Canada
iaa7@cs.sfu.ca

Abstract

We present a crowd-sourced service that enable users to experience large social events with the focus on the context of the captured media. In order to enable such a rich experience, a centralized service needs to organize and annotate a large number of media files including photos, videos, and live streams. Our cloud based service receives user generated content and construct a 3D representation of the physical world. The system uses structure from motion algorithms to estimate camera poses of the contributed media. This information is used in a 3D media viewer that allows users to explore many images and videos in their original context. We demonstrate the potential of such a service with a proof-of-concept implementation applied on a small dataset of media items.

I. Introduction

A number of user-generated media sharing web services have emerged in recent years. An early prominent example of these services is the Flickr image and video hosting website [1]. The ability to publicly share user media have also expanded to most social networking sites such as Facebook, Twitter, and Instagram all of which have millions of active users. Most of these services organize their media based on authors, album collections, playlists, or user assigned tags. Other services, such as Google's Panoramio and 360Cities, focus on geotagged content that relates to physical locations and so organize their media spatially. All of these services allow users to explore past events after the media is collected and organized, thus losing the excitement and appeal associated with live events. Current live streaming services such as Ustream and Google Hangout allow users to engage and communicate with each other typically regarding a single video stream. This trend of live user interactions is also translating to mainstream TV, for example the iDesk Hockey Night on CBC Sports [2].

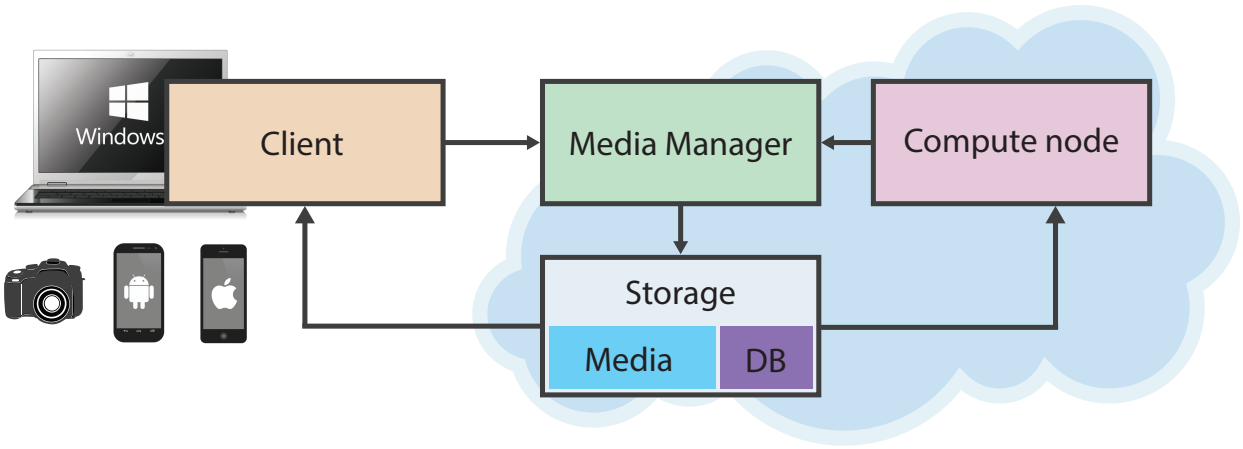

Fig. 1. Overview of the Eventful Cloud Service. For more details, please see the text.

We present a crowd-sourced live events presentation service that focuses on providing a more immersive viewing experience. The service combines images, videos, and live streams from different sources and organize them based on location and context. Each media entry is processed only if it satisfies a level of accurate information of time of capture, location, or both. During a live event, a progressively generated 3D representation help users explore large amounts of submitted media in their original context shortly after their acquisition. Our main goal is to democratize such a computationally expensive task that is currently limited to research implementations or exclusive to specific applications in industry.

Our system is composed of four main components: the *client*, the *media manager*, the *compute node*, and the *storage unit*. The client is a device capable of capturing or uploading different media items to the web. It can be either active or passive depending on its ability to capture or display media items. The media manager is responsible for receiving, preprocessing media, and directing requests to the appropriate storage unit. The compute node is responsible for executing the computationally expensive operations that progressively reconstruct a 3D representation of the live event. The system design focuses on making the different components as light-weight and portable as possible. This allows the system to be highly scalable and work efficiently depending on the demand on both the client and server side. Figure 1 shows an overview of the system's components.

We provide a proof-of-concept implementation that is currently deployed on the Amazon EC2 cloud computing platform. Our experimental results demonstrate several use cases for live events captured and viewed by different clients using different devices.

In the following we will start by describing some of the related work (Section II) in the field of

photo exploration and real world 3D reconstruction. We then detail the structure and design goals of our proposed system (Section III). We also show results obtained from our basic implementation of the system (Section IV) and then conclude the paper (Section V) with an outlook on the future.

II. Related Work

Several studies have been conducted regarding the way people manage their collections of photographs. The study of [3] showed that people expect from photo management tools a basic organization feature that automatically sort photos in chronological order and show as many thumbnails at once as possible. Other important aspects they found is that people are not generally willing to provide textual annotation to their photographs. Others effort [4] were focused on automating aspects of the process of organizing photos by their time and content. This approach could help users explore a large image collection with some level of context during an event. However, the lack of semantic information of the photographs would result in grouping visually similar images that are not related.

The survey [5] explores many methods relating to image retrieval and automatic annotation. Issues relating to the problem of photo collection exploration include: annotating a large number of items, figuring out users viewing intent, defining a search scope, and defining and evaluating a usefulness metric of the different systems. The survey suggest that future photo organization and exploration systems will need to incorporate advances in different fields such as multimedia, machine learning, computer vision, and human-computer iteration.

The *photo explorer* system is a successful effort in exploring large collections of photographs. The system applies different advances in computer vision and *structure from motion* (SfM) allowing for a user-friendly 3D exploration interface [6]. The system uses thousands of uncalibrated photos taken using different sources and extract camera poses and sparse 3D scene information that is used in their viewer. Such extracted sparse points can then be used to generate dense 3D reconstructions of the scene. In [7] a patch-based multi-view stereo method generates detailed 3D structures from a small number of 2D photographs. The method works by gradually matching then expanding on a sparse set of initial points. The resulting 3D geometries can be highly detailed, however, the technique comes with a large computational cost. In order to apply this method on extremely large photo collections, the work by [8] focuses on decomposing the input into a set of overlapping sets that can be reconstructed in parallel. Their dataset consisted of tens of thousands of photographs that resulted in millions of 3D points representing buildings and further improved to allow for entire cities [9]. These techniques are now used in industry, for example in Google Maps [10], to generate static 3D reconstructions of different landmarks around

the world. In our system we used a simplified pipeline similar to these methods using the implementation provided by Noah Snavely [11] and Yasutaka Furukawa [12].

Different frameworks have recently been proposed for computation offloading from limited devices to the cloud [13]. The current trend for mobile based user applications is to model systems where a mobile device is a front-end user interface for a complex and computationally expensive service hosted elsewhere. The mobile device also serves as a multipurpose sensor dispatching commands, photos, sounds, and location to a more capable infrastructure using the web. Also, having the user's media stored in the cloud would allow for larger media collections than what is available on the capturing device. The survey [13] listed many application that make use of mobile cloud computing including image processing and OCR, natural language processing, music identification, and many other multimedia applications. Proposed methods of offloading include client-server communication, virtualization, and mobile agents. In our system design we follow the client-server approach with the emphasis on portability. Our implementation uses modern web standards that allow for rich web applications. Using web capabilities ensures accessibility to the service from a wide range of devices, i.e. any device with a browser that is HTML5 ready. For the server side we use a typical virtual machine in a cloud computing platform.

III. Proposed System

Our proposed system is composed of four separate components: the client, the media manager, the compute node, and the storage unit. The interaction between these components is shown in Figure 1. The client provides an interface for users to create and view live and past events. It interacts with the service through a simple web API provided by the media manager. The media manager listens for requests from both the client and the compute node and responds with direct links to the storage unit. The compute node is an abstraction of one or more virtual machines that is specifically setup to perform a single computationally expensive task. In our service the task is 3D scene reconstruction and photo registration.

During an ongoing event, different users will share and upload the different media items for storage and processing. Once the media is uploaded it is immediately available for both viewing and processing. The compute node periodically asks the media manager for jobs still waiting in a queue. Once a job is done it is sent to storage and becomes readily available for clients. The goal of this modular design is to minimize overhead and allow for extensibility.

In the following we detail the different components of the system. We first define the conceptual ideas behind each component and then describe implementation details in our proof-of-concept realization of the system.

A. The client

The client is any connected device that is capable of sending or receiving media using the web. It provides an interface for users to create, view, and explore social events that are defined by a time and a physical location. Events can range from private gatherings to international sport events. Important characteristics for the client is responsiveness and having an intuitive interface.

Different users can have different access privileges for each event. For example, organizers of a city wide festival would designate event admins that are able to post announcements and sponsored messages to be displayed to regular viewers. The timing and location of the announcement would help focus the event attendees to a particular booth or easily announce reminders about major activities. Other privileges can give access to security personals to help in safety or possibly locate a missing child. A commercial opportunity is to provide VIP access to viewers, for example behind the scene interviews, at a premium. For regular small social gatherings, such as personal parties, viewers only need a passkey to be able to view and contribute to the event.

There are two event viewing modes available in the client: the *photo stream view* and the *context view*. Figure 2 shows the two modes in action. The photo stream view is the default view that presents the user with large thumbnails of the event's media sorted into chronological order. This view presents a familiar interface commonly used in current photo hosting services with some added functionality that are useful for live events. Media items are displayed as a 2D grid with rows representing the event's time and columns representing multiple media items captured at the same time frame. Photo and video items are navigated by swiping on the photo stream horizontally with newer items appearing at the right most column. Media items in the same column are sorted based on the current distance between the location of the client and the location from where the media item was captured from.

The second viewing mode is the context view mode. In this mode the users are presented with a 3D reconstruction of certain parts of the physical location of the event. The users can navigate through a large number of captured images contributed by attendees logged into the same event. Newly registered media are presented with visual queues such as color contrast or fading of older items. This allows users to focus on the most current and active parts during an event. Part of the context view is the map view where media items are plotted on a physical map of the event's location. For media items not captured with GPS coordinates, the 3D registration of the item can help provide a relative approximation of its geolocation.

We've implemented a simplified version of the client using only HTML and JavaScript code. Viewing

images is an integral part of all web browsers and so no special technology is needed. For videos we use the HTML5 video element currently supported by both desktop and mobile browsers. Since most modern mobile devices and cameras support the H.264/MPEG-4 AVC codec, we assume that all video items will be stored using this widely used codec; if needed we convert the video to it. For the context view we use the three.js JavaScript library [14] to render the 3D scene having the reconstructed geometries and camera poses. This library uses the WebGL capabilities of modern browsers and if no support is available, for example on iOS devices, it will fallback to using the HTML5 canvas renderer. Our implementation uses the WebRTC API to capture photos and videos on the desktop and uses the PhoneGap framework [15] for mobile devices when available. For retrieving the location of captured media we relay either on embedded GPS coordinates recorded by modern cameras or simply use the HTML5 geolocation API when the user uploads the media. For the event's map view we use the Google Maps API in order to retrieve real world maps of the event.

During an active event the input to the client, generated by querying the media manager, is an organized list of media items with direct links to the storage unit. The output to the media manager from the client are media data streams with their geolocation when available.

B. Media manager

The media manager is a server actively listening for client and compute nodes requests. For client requests, it is responsible for user access operations, the creation and management of events, and most importantly media management. Each media item received from a client is preprocessed and sent in its final form to the storage unit. Preprocessing may include: automatic extraction of EXIF tags, generating photo and video thumbnails of different sizes, rating media content based on some specified metric, re-encoding videos to different formats, filtering copyrighted and inappropriate content, or simply avoiding duplicates. This component is also responsible for fetching detailed information of a particular media item selected by the user. Also, the media manager is capable of serving different representations of the media based on the client's request. For example, when a new client requests a media in a different size or codec the manager would dispatch a media processing event and returns the result to the client. For compute node requests, the media manager would look for and generate job descriptions for active events. For each event it compiles a list of not yet registered media items and sends a response to the requesting compute node. The decision to accept the job is decided by the compute node as described later on.

In our current system, we implemented a simplified version of the media manager. Our media manager

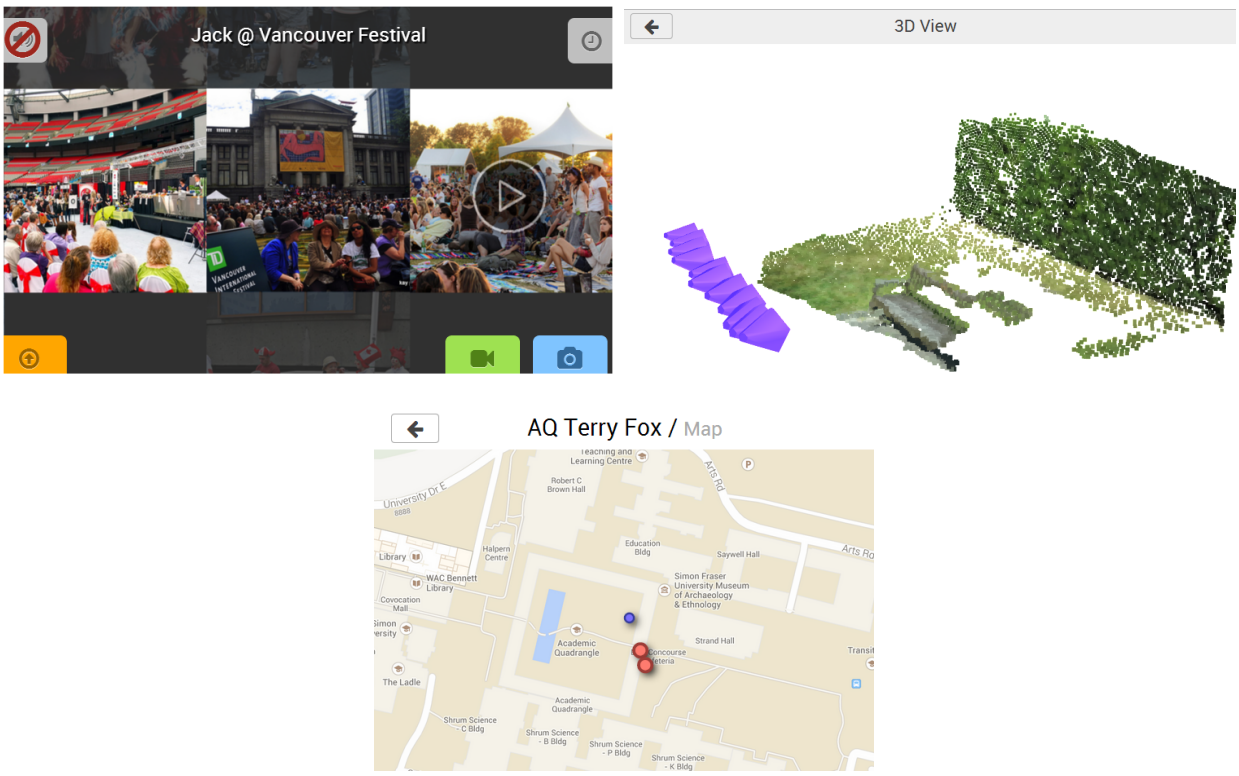

Fig. 2. The two main different client views available in our system (top). The photo stream view (left) with both photos and videos. The 3D context view mode (right) of the Bench dataset where the purple shapes represent the registered camera positions. A map view is also available for GPS registered media (bottom).

implementation consists of a couple of PHP scripts hosted on an Apache HTTP Server. The PHP scripts call the open source ImageMagick library for image manipulation and processing, and the FFmpeg software when processing video items. Other processing or filtering mechanisms can be easily attached to the pipeline when needed.

Major advantages of having a simple HTTP based server is the ability of the manager to scale based on demand and move across different hosts. Since Apache servers have proven to be able to handle heavy traffic, we believe that creating large numbers of VM instances on the cloud allows for a scalable solution. The only requirement of a media manger is the ability to track the locations of the storage units. One approach is to designate a fast and highly reliable service that is only used to resolve these locations. Also, since the media manager is written using simple PHP scripts and cross-platform open-source libraries it is highly portable. Being highly portable allows the manager to live on different machines at the same time, for example on edge servers that are physically close to the live event or even to the client device itself.

All mentioned tools above are currently available on the Android platform including a PHP interpreter. This can result in network traffic savings for mobile devices but at the expense of power consumption. Moving the media manager to the client side would also allow for interesting caching opportunities that we have not yet explored.

C. Compute node

The compute node is an abstraction of a single machine or a network of machines that is setup to execute a specific computationally expensive task as many times as possible. The task required in our system is the structure-from motion (SfM) processing of a large number of unordered images. There are four stages in the SfM pipeline used in our system: data collection and pre-processing, 3D reconstruction of camera poses, clustering views into sets, and dense scene reconstruction.

The first stage is data collection and pre-processing. Since the event's media may live outside the compute node, a pull request to the media manager is first performed. The request asks the media manager for the next set of jobs as decided by the queuing strategy. Possible queuing strategies include priorities based on the event's size, sponsorship, current state of reconstruction or other conditions that depend on the service providers business model. A simple round-robin scheduling could always serve as a default mechanism. The pre-processing of the input data can help speed up the reconstruction process and potentially increase the quality of the output. The most basic operation is scaling down large images in early reconstruction iterations that results in large speed ups. Another possible processing is discarding blurry images, noisy or dark images, images below a color or contrast threshold, or personal images with relatively large human faces, identified using basic face detection methods, that do not contribute much to the scene. For video items, we sample the clip by extracting frames at the rate of 1 frame per-second and then treat them as we do with still images.

The second stage in the pipeline is the 3D reconstruction of both camera poses and sparse geometry of the scene. This tasks is accomplished by running a SfM software that takes an input of unorganized images and outputs a basic reconstruction. The process itself starts by computing a set of image features that are typically obtained using the scale-invariant feature transform (SIFT) algorithm [16]. The next step in the process is finding image matches by comparing keypoint descriptors of each pair in the input images. The result of these steps is then used in an optimization process that reconstructs the scene incrementally as detailed in [6]. The SfM software outputs computed camera information for registered images, such as 3D position and rotation, and a set of sparse 3D points representing parts of the scene with position and color information. This output is then immediately available for viewing and further

Fig. 3. Result of the dense scene reconstruction on the Terry Fox dataset.

processing. A useful final step in this stage is to perform geo-registration of the reconstructed scene to match with a reference map of the event. This process can be automated for outdoor scenes with the availability of GPS coordinates in several registered images.

The third stage in the pipeline is the view clustering stage. The Clustering Views for Multi-view Stereo (CMVS) method [8] decomposes the problem of multi-view stereo in order to make it more manageable. The key idea in this method is to divide the input image collection into a set of image clusters that can fit into memory. This decomposition helps in speeding up the scene reconstruction by processing the different clusters in parallel and identifying and discarding redundant images. This step is especially useful for compute nodes composed of a network of connected machines.

The final stage is the dense 3D scene reconstruction. The Patch-based Multi-view Stereo method [7] takes as input the resulting sets from the third stage and outputs a dense set of 3D points with location and color information. The algorthim works by first detecting image features, such as corners, and matching them across the set. Next, the matched regions are expended iteratively by adding more points with the help of previous matches. The final step is a filtering process that removes any outlines based on visibility consistency measure. The full details of the method can be found in the original paper [7]. Typically, hundreds of images result in millions of reconstructed 3D points. Figure 3 shows an example of the dense reconstruction process.

In our implementation we have used the source code available by the original authors of the different methods used in the reconstruction process. These include the Bundler, CMVS, and PMVS packages. The most time consuming part is the dense reconstruction done by the PMVS software. Therefore, we designed the system to export the results of the sparse reconstruction, i.e. the output of Bundler, as soon as they are ready and make them available in the client view. The PMVS process would then start and once it is finished the scene viewed in the client will eventually become richer and more detailed. The packages also support incremental addition of new images which can greatly reduce the time needed for registering new media items. We report on timing in the results section.

There are several opportunities of optimization for the compute node component. Since the setup is composed of compiled executable code it is highly portable. If the media manager takes responsibility of launching compute node instances, it can assign new nodes closer to where the media resides. For example, the Amazon cloud computing platform have multiple data centers across the world. For an event that is physically happening on the west coast it can be stored and processed on the servers located in the Amazon Oregon data center. This would dramatically decrease transfer speeds and network costs on the server provider.

Also, gradual reconstruction is another opportunity to increase responsiveness of the system and the overall viewing experience. Since our system focus on the viewing aspect of the reconstruction process, we can modify the reconstruction parameters to match the demand on the system. During heavy load in a particular region, the PMVS dense reconstruction process should not execute until all sparse reconstruction tasks, Bundler tasks, are finished. Other factors such as the number of active participants or sponsorship status can also decide on the level and quality of the requested reconstruction.

D. Storage unit

Media hosting services such as Flickr deal with a large and increasing number of uploaded content each year [17]. Their architecture is composed of many stacks of app and caching servers, storage managers, and a search engine infrastructure [18]. The reason for having such a complex infrastructure is the need to efficiently handle millions of media requests each day. There are several lessons to be learned from existing services regarding scalability. However, we will assume that the existing storage services provided by cloud services such as Amazon are sufficient for our application.

We choose to use the Amazon cloud storage service which have proven capable of handling high file transfer demands as demonstrated by hosted services such as Dropbox, SmugMug, and Instagram. The Amazon file storage services have different layers each having a different performance depending on

the usage. The Amazon Simple Storage Service (Amazon S3) is the standard layer that is widely used by Amazon cloud service clients. The fastest layer is the CloudFront service that work on making data available to edge locations on the Amazon network. The cheapest and slowest service is the Amazon Glacier service geared toward data archiving.

Uploaded media in our system are stored in the Amazon S3 file storage service accessible by a predefined URL address. This allows our client and compute node components to easily find and query an event's media. However, a database is still needed for users information, event details, and tracking each item's 3D registration status. That is why our storage unit should also have a basic database component. In our implementation we used a MySQL database hosted on the Amazon Relational Database Service (RDS). The Amazon RDS service simplifies many database tasks such as creation, scaling, and recovery. Since our system's database structure is simple, it may be replaced with a NoSQL mechanism which might be useful if data retrieval performance becomes an issue.

IV. Evaluation

In the following we describe our proof-of-concept implementation using a small dataset and present results obtained using our setup. We also show some promising results when comparing our implementation to a highly parallel implementation. We then outline some of the limitations of our work. Our implementation is currently available at this URL https://54.214.248.120.

A. Implementation

Our initial development setup consisted for a single local machine with an Apache HTTP server hosting a MySQL server. As development progressed, we migrated the server code to the Amazon EC2 cloud computing framework. We used the most basic instance available, the M1 Small Instance, having 1.7 GB of memory and 1 virtual core with 1 EC2 Compute Unit. This type of instance is provided free of charge for new clients on the Amazon network. We then divided the server responsibilities into two instances where one is responsible for the media manager component and the other is used for reconstruction computations. Migrating the MySQL database from the virtual instance into the Amazon RDS helped speed up development since it ensures that the development and production machines used the same database. Also using an HTML based client made it possible to test the system on many different devices. Our client tests consisted of: a 3.0 Ghz desktop machine with 4.0 GB of memory, a Samsung Galaxy S II (SGS2) Android smart phone, an iPhone 4S phone, and the Android based Asus Eee Pad Transformer TF101 tablet.

TABLE I

DATA SETS AND STATISTICS. HEIGHT IS MEASURED IN PIXELS. TIMING SHOWN IS IN MINUTES:SECONDS.

Name	Num. photos	Height	Bundler	PMVS
Terry Fox	40	200	3:35	6:54
Waterfall	4	300	0:14	0:48
Bench	14	400	14:58	19:23
Juice store	20	300	1:16	4:30
Lobby	12	300	1:40	16:02

B. Results

We collected several images and videos around the Simon Fraser University Burnaby campus. We used two sources to capture our data: the SGS2, a GPS-equipped phone, and a Sony NEX-5N camera. Our photo sets represent small scenes with the smallest one captured with 4 images and the the largest using 40 images. We downsample our input images to a size of 400 x 300 pixels in order to operate at a reasonable rate of couple of minutes per event. This results in lower resolution reconstructions, however, the focus of our application is on navigation rather than accurate scene reconstruction. The number of densely reconstructed points in our test collections on average is 5000 3D points.

Table I shows an example of the data sets used in our experiments. For the Terry Fox dataset we downsampled the images to ones with height 200px in order to compensate for the larger number of photos in the set. The Waterfall set demonstrate reconstruction with large changes between each frame. The Lobby dataset represent an indoor scene under fluorescent light. In all of the results, only partial reconstructions are possible. This is due to the limitation of SfM methods requiring a large number of different views. Figure 4 shows some of the reconstructed scenes from our dataset.

The results show that the PMVS process takes the most amount of time. We believe that this component can benefit greatly from parallelization. Unfortunately, this would require a more complex design of the system and a comprehensive rewrite of the original source code. This is an active research problem with some suggested solutions from the authors of the tools we have used [19]. In its current state, our best solution for the slow reconstruction is to reconstruct at lower resolutions. With more media added during a live event, the quality of the reconstruction would allow for a reasonable 3D viewing experience. Another possible solution is to rent faster instances on the Amazon platform that are only used for the initial reconstruction stages of the event.

In order to show the advantages of a parallel implementation we compare the total timing between

our results and a faster GPU-enabled implementation from [20] on a Windows desktop machine. The following table shows results of reconstructing two of our datasets with the same parameters on the different machines:

TABLE II
AMAZON INSTANCE VS. GPU IMPLEMENTATION.

Name	VM time (s)	GPU time (s)
Waterfall	62	8
Juice store	346	48

C. Live media streaming

We have experimented with a couple of options for live media streaming. We found that a dedicated host still remains the best option for such a feature, especially for cross-platform support. An example of this is the Zencoder cloud-based video encoding service or the Amazon Elastic Transcoder service. Live streams can be recorded and archived during an event. This can be helpful as the recorded stream can be sampled and incorporated into the reconstruction process as well. However, having a static camera position does not necessarily help with the problem of missing scene information.

D. Limitations

Our implementation represents a bare minimum setup for our proposed Eventful Cloud service. The results only demonstrates a working pipeline from capturing the media to viewing it in context. However, the datasets used do not represent a real event scenario where possibly hundreds of people contributing large number of media items at the same time. Also, media items registered are not yet set to their real geolocations. Only GPS tagged media items currently show on the event's map. Other essential missing functionalities of our implementation include: event reconstruction scheduling, media filtering, and the lack of any caching mechanism.

We have noticed a couple of issues regarding the user experience of our system. The first issue regarding the viewing experience is the problem of visual clutter. Having many images for the same region doesn't contribute to an exciting experience. We believe that a clustering and random sampling process of similar items would help mitigate this problem. Another possible problem is finding the balance between displaying the user's personal media with family or friends and media contributed by other attendees. A user study for the overall user experience is definitely needed to evaluate such issues.

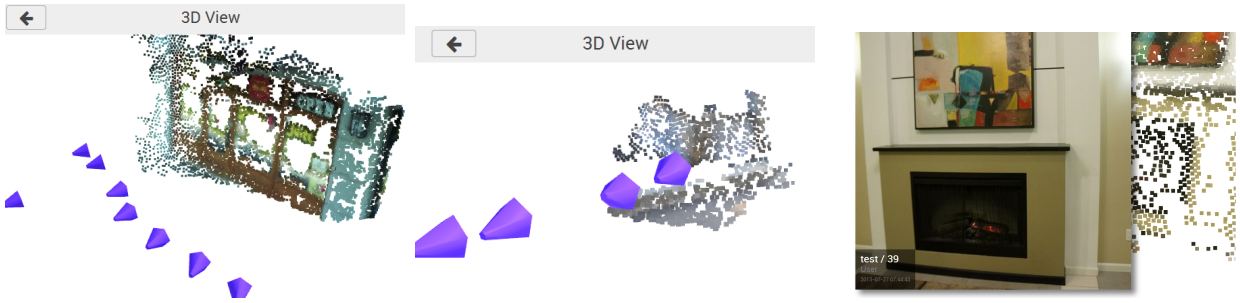

Fig. 4. Examples of the reconstructed scenes as viewed in the client: the Juice Store (left), the Waterfall (middle). The purple cones represent the registered camera positions. The Lobby example (right) shows the actual image when the user clicks on one of the registered cameras.

V. Conclusions

We have proposed a crowd-sourced social event viewing service. The system applies advanced computer vision techniques in order to embed media items in a 3D scene that would reflect how the media was captured and in what context. The goal of the system is to create a shared immersive experience of the a live social event. We explored how a cloud based service can allow a limited device, such as a camera or a smart phone, to create rich multimedia scenes. Our experience with the Amazon EC2 cloud computing platform showed us that such such platforms allow for a short development cycle. We believe this is a promising outlook on the future of cloud computing services.

There are still many technical challenges before achieving the goals of our proposed system. The reconstruction process is a computationally expensive task by nature and so we still have to compromise on quality and speed. Some promising results of parallelization and other advancements on the original methods can help resolve some of these issues. However, we believe that any solution needs to take into account several important logistical issues such as operational costs and event priories.

VI. Future work

The reconstruction process could make use of third party data such as Flickr or Google Street View to help increase the quality of the reconstructions by filling in missing data. Another idea is embedding the current reconstructions into the camera application and providing visual feedback on where missing data needs to be filled. Also, indoor positioning systems [21] is an active research field with promising results. We believe that being able to estimate relative positions of media can help with clustering related views more efficiently resulting in shorter reconstruction times with higher quality. Incorporating such technologies may attract more users to use a service like the one proposed.

REFERENCES

[1] Flickr, www.flickr.com, 2004.

[2] CBCSports, "Hockey night in canada live video and chat," www.cbc.ca/sports/hockeynightincanada/video/live, 2010.

[3] K. Rodden and K. R. Wood, "How do people manage their digital photographs?" in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '03. New York, NY, USA: ACM, 2003, pp. 409–416. [Online]. Available: http://doi.acm.org/10.1145/642611.642682

[4] M. Cooper, J. Foote, A. Girgensohn, and L. Wilcox, "Temporal event clustering for digital photo collections," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 1, no. 3, pp. 269–288, Aug. 2005. [Online]. Available: http://doi.acm.org/10.1145/1083314.1083317

[5] R. Datta, D. Joshi, J. Li, and J. Z. Wang, "Image retrieval: Ideas, influences, and trends of the new age," *ACM Comput. Surv.*, vol. 40, no. 2, pp. 5:1–5:60, May 2008. [Online]. Available: http://doi.acm.org/10.1145/1348246.1348248

[6] N. Snavely, S. M. Seitz, and R. Szeliski, "Photo tourism: exploring photo collections in 3d," *ACM Trans. Graph.*, vol. 25, no. 3, pp. 835–846, July 2006. [Online]. Available: http://doi.acm.org/10.1145/1141911.1141964

[7] Y. Furukawa and J. Ponce, "Accurate, dense, and robust multi-view stereopsis," in *Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on*, 2007, pp. 1–8.

[8] Y. Furukawa, B. Curless, S. Seitz, and R. Szeliski, "Towards internet-scale multi-view stereo," in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, 2010, pp. 1434–1441.

[9] S. Agarwal, Y. Furukawa, N. Snavely, I. Simon, B. Curless, S. M. Seitz, and R. Szeliski, "Building rome in a day," *Commun. ACM*, vol. 54, no. 10, pp. 105–112, Oct. 2011. [Online]. Available: http://doi.acm.org/10.1145/2001269.2001293

[10] G. Maps, "Visit global landmarks with photo tours in google maps," http://google-latlong.blogspot.ca/2012/04/visit-global-landmarks-with-photo-tours.html, 2012.

[11] N. Snavely. (2013, July) Bundler: Structure from motion (sfm) for unordered image collections. Cornell University. http://www.cs.cornell.edu/ snavely/bundler/.

[12] Y. Furukawa and J. Ponce. (2013, July) Patch-based multi-view stereo software. Ecole normale superieure. http://www.di.ens.fr/pmvs/.

[13] N. Fernando, S. W. Loke, and W. Rahayu, "Mobile cloud computing: A survey," *Future Gener. Comput. Syst.*, vol. 29, no. 1, pp. 84–106, Jan. 2013. [Online]. Available: http://dx.doi.org/10.1016/j.future.2012.05.023

[14] R. Cabello, "three.js javascript 3d library," http://threejs.org/, July 2013.

[15] Adobe, "Phonegap," http://phonegap.com/, July 2013.

[16] D. Lowe, "Object recognition from local scale-invariant features," in *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, vol. 2, 1999, pp. 1150–1157 vol.2.

[17] K. Kremerskothen, "Six billion," http://blog.flickr.net/en/2011/08/04/6000000000/, August 2011.

[18] highscalability.com, "Flickr architecture," http://highscalability.com/flickr-architecture, November 2007.

[19] D. Crandall, A. Owens, N. Snavely, and D. Huttenlocher, "Discrete-continuous optimization for large-scale structure from motion," in *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, 2011, pp. 3001–3008.

[20] C. Wu, "Visualsfm : A visual structure from motion system," http://homes.cs.washington.edu/ ccwu/vsfm/, 2012.

[21] K. Al Nuaimi and H. Kamel, "A survey of indoor positioning systems and algorithms," in *Innovations in Information Technology (IIT), 2011 International Conference on*, 2011, pp. 185–190.