2010 Second International Conference on Evolving Internet

A Distributed Content Storage Model for Web Applications

Mark Wallis, Frans Henskens, Michael Hannaford
Distributed Computing Research Group
University of Newcastle
Newcastle, Australia
Email: mark.wallis@uon.edu.au

***Abstract*—Vast quantities of information is now being stored online. Web applications currently rely on monolithic storage structures which place the sole responsibility of data storage, protection and maintenance on the web application provider. This research introduces the concept of a de-centralised approach for information storage online. Distributed storage techniques are used to address concerns with the classic monolithic approach while also addressing issues such as data ownership concerns for personal information. The research results in the presentation on an API that allows distributed storage of information with seamless integration of data into the traditional Web 2.0 model.**

***Keywords*-distributed; storage; personal data; ownership.**

I. Introduction

The current breed of Web Applications, known as Web 2.0 [1], rely on monolithic storage models which place the sole burden of data management on the web application provider. Having the application provider manage this data has resulted in problems with data ownership issues, data freshness and data duplication, as has been covered by our previous work [2]. As the majority of data in Web 2.0 is user-generated, it is suggested by this paper that the responsibility for storing user-generated content should be given to the data owner themselves. As such, this research presents a distributed storage model, which allows web applications to offload the storage and management of user-generated content to storage systems managed by the data owner. The Distributed Data Service (DDS) API allows web applications to seamlessly present user-generated content to a 3rd party user. The 3rd-party interacts directly with both the web application and numerous DDS systems which host the web application's content. Data owners can manage their data elements by interacting with a DDS directly while also exposing this data to web applications via a publish/subscribe model.

The justification behind a distributed storage model is based on the concept that in a Web 2.0 application the majority of information is generated in a distributed fashion by end users. The term 'Web 2.0' is a business generated term, which can be traced back to 2005 when O'Reilly first defined the concept of web generations [1]. At the time, Web 2.0 was identified as any web application that matched the following criteria:

- The application represents a service offering and is not pre-packaged software.
- The application data evolves as the service is used. This is in contrast to applications in which static data is generated solely by the application owner.
- A framework is provided that supports and encourages user submission of software enhancements. These submissions are generally in the form of plug-ins or extensions to the web application.
- Evolution of the application is driven by the end user as well as the application owner.
- The application interface supports interaction from multiple client devices such as mobile phones and PDAs.
- The application provides a lightweight, yet dynamic, user interface.

A high-level overview of the Web 2.0 design is shown in figure 1.

A key criterion of interest to this research is that evolution of the website is tied to the degree of user interaction. This is driven directly by the fact that the primary service provided by a Web 2.0 site generally relies heavily on user-generated content. The more that data owners interact with the website, the greater the experience of all users. This paradigm has proven popular because website owners are no longer solely responsible for content generation. The fact that the amount and richness of the data provided by data owners can be directly tied to the success of a website only exacerbates the problem of data ownership, as the contained data becomes an important asset for the website owners. Without this asset, they would have substantially less to offer to their user base.

This paper formally defines the interface used by the distributed data service - the DDS API. Section II follows with an overview of the problems which the DSS addresses. Section III formally defines the DSS API as a specification that can be used to implement distributed data systems using a distributed content management model. It encompasses all phases of the data management including the insertion and retrieval of data by 3rd parties. Section IV describes the proof-of-concept implementation provided by this paper which includes multi-language and multi-platform components. Sections V and V-C present performance metrics and provide a comparison of the DSS solution against other existing technologies. Section VI provides an overview of how the solution scales before Section VII presents an overview and conclusion.

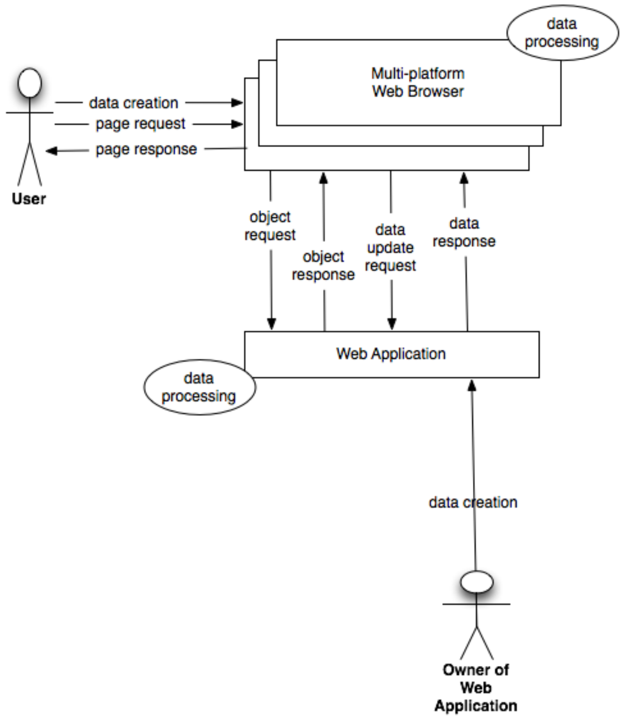

Figure 1. Web 2.0 model

II. Problem Description

The problems with the existing storage models in Web 2.0 can be viewed from two aspects - the web application and the end-user. From a web application viewpoint, monolithic storage structures have resulted in the following key problems:

1) Web applications must host and manage large-scale database systems to store all the user-generated content in their data centre.
2) Since all the data is stored by the web application the hosting network must bear the full load of transferring that data between the monolithic storage structure and the user-base.
3) Web application providers must deal with complex privacy and regulation issues stemming from the fact that access to user-generated content can sometimes be restricted by privacy laws.
4) Monolithic storage models introduce large single-points-of-failure. In a Web 2.0 model, the risk is that a single Web application hosting valuable data may go offline, either temporarily or permanently.

Three key problems affecting the end-user are data ownership, data freshness and data duplication [2].

1) Data ownership is the most important issue of the three. This relates to the fact that 3rd party web applications can currently place restrictions on the usage of a user's data just because they store it locally on their servers. Data owners are forced to agree to EULAs to use services that can take away the owner's rights to their own data, despite them owning the original data [3].
2) Data freshness refers to the situation in which a user provides data to one or more web applications and that data changes at a later date. The original data provided to the web applications then becomes stale, unless the user is able to recall and update all web applications to whom the data has been provided. The manual user update would be performed on a per-web-application basis, possibly dealing with access issues such as expired login or forgotten credentials.
3) Data duplication refers to the situation in which multiple web applications store copies of the same piece of data. While the associated implementations seems trivial when considering such pieces of data as a single postal address, the issue expands dramatically when dealing with multimedia data such as image, music and video libraries.

These issues have all developed through the increased use of SAAS and Web 2.0 architectures. SAAS (Software as a Service) is a model where software is provided as an online service as opposed to a distributed executable piece of code. While the benefits of a SAAS model are well documented, it is the combined use of SAAS with the increased level of user-content being stored online which has led to the above problems. The model presented in the remainder of this paper resolves these issues while also maintaining the benefits of SAAS and the Web 2.0 model.

III. Model: DDS API

The model presented in figures 2 through 6 addresses the identified problems by introducing additional technology that allows storage of data to be offloaded from the web application. Instead, the storage is made the responsibility of 3rd party storage providers. These providers lease storage services to individual data owners, and act as a 'single version of the truth' provider for that piece of data. Data-owners can either be corporations, business groups, public entities or even individuals. Enhancements to the standard web browser design allows this data to be accessed seamlessly for integration into displayed web pages. An API is established to govern communication between the various actors in the model. These web browser enhancements are a stepping stone between existing browser technologies and a complete Super-Browser implementation [4].

The model definition can be broken into three main areas: storage, access and presentation.

A. Storage

The first phase of distributing data storage involves the data owner subscribing to a distributed data service (DDS). This service is responsible for storing the owner's data elements and is located either in-house or outsourced to a specialised data storage provider. The subscription procedure is shown in Figure 2.

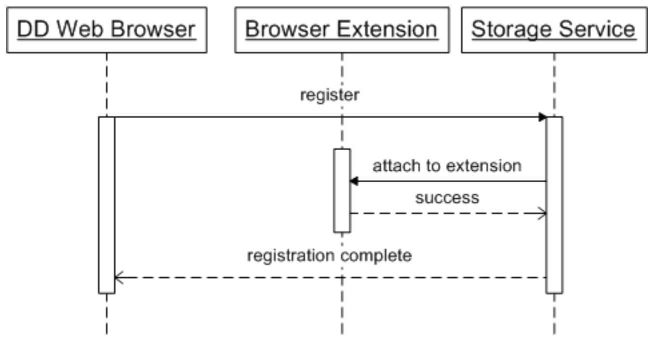

Figure 2. SS Registration

To aid integration of the DDS into the data owner's web experience a new module is introduced called the 'DDS browser extension'. This extension executes within the data owner's web browser, and is aware of the new distributed data service model. When a Web 2.0 site requests content from the data owner, he or she provides a link into the DSS in which the data is stored. Later, when the data is needed for display as part of a page generated by the site, the displaying browser instance uses the embedded link to directly retrieve the data from the owners's DSS. In current pilot implementations of the model, the module is implemented using the browser-extension technologies provided by most modern web browsers [5]. Future Super-Browser implementations will see the module as a distinct component executing within the web browser virtual machine [4]. Communication between the browser extension and the storage service is achieved by inserting a *DDS-StorageService-AttachRequest* into the HTTP response. This request is transparently inspected on-the-fly by the browser extension which allows the extension to re-write parts of the response HTML dynamically. For the data-owner, the extension re-writes HTML form fields with links that activate a browsing interface for selecting data objects. For the end-user, the extension re-writes links into a remote DSS with data returned from that specific DSS.

Implementation of the storage service registration process and the browser extension is implementation specific, though the API for linking the two components together is the first aspect defined by the DDS API. As long as implementations maintain the API for the DDS series of request/response messages, interoperability is maintained. The security of the connection between the browser extension and storage service is also implementation specific, but it is assumed that SSL encryption would be used at the tunnel level and basic authentication would be used to authenticate the end user to the storage service.

Phase two of the process, presented in figure 3, involves the data owner publishing content to the storage service. Again, the implementation is not restricted by the API as long as each piece of stored data is given a unique identifier that is global in that data owner's domain. The unique identifier comprises three components:

[name]:[path]@[system]

The *name* component represents an identifier for each specific piece of data (for example, credit_card). The *path* component supports a hierarchical storage structure allowing a data owner wishing to store various groupings of data (for example, a data owner may have separate sets of 'personal' and 'business' data). The *system* component is a unique identifier for the specific distributed data service. It is generally a DNS name referencing the DDS itself. To reduce vendor lock-in it is recommended that data owners implement their own DNS pointers so that migration from one DDS to another does not result in the reissue of unique identifiers due to a change in the related system component. Using DNS for the system component also solves the issues of locating a specific entities DSS.

The data format proposed by this design is based on a published set of XML schemas that represent each type of data stored by the DDS. While enforcing data format appears restrictive, it is necessary to ensure that interoperability is promoted between web applications and storage services. Such generic operability is one of the main requirements for a solution that does not promote vendor lock-in.

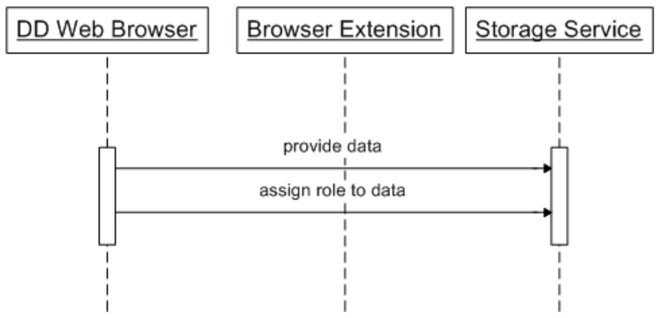

Figure 3. SS Information Upload

B. Access

Once the data owner has uploaded data into their DDS, the next stage is to allow web applications to subscribe to this data. This is presented in figure 4. During the registration process for a DDS-enabled web application, the web browser extension inspects the HTTP response traffic from the application and detects a *DDS-Application-Subscribe-AuthRequest* request. This request, and the corresponding response generated by the browser extension, is the basis for establishing a trust relationship between the web application and the distributed data service. The request/response messages are built on extensions to SAML [6] and act as a basis for exchanging PKI credentials between the end user and the web application.

Once a relationship is established, as shown in figure 4, the data owner establishes a link between their data element and the web application. This link is akin to the data owner uploading content to the web application in a standard Web 2.0 scenario, except that in the DDS design the data owner provides the unique identifier of the data as opposed to uploading the content itself.

The browser extension again plays a key role, ensuring that web applications can support both DDS-enabled and classic Web 2.0 clients. Additional DDS-enabled attributes are inserted into HTML *input* tags to inform the browser extension that the data owner's interface should be modified to accept a unique ID value rather than actual data. This linkage maybe optionally implemented by showing a pop-up window containing an index of the data stored in the DDS to allow the data owner to select individual pieces of data graphically, rather than manually entering the unique ID of the data.

Once the link is established between a piece of data referenced by the web application and the storage location for that data in a DDS, the web application is free to request the data directly from the DDS. This is useful in the circumstance where the web application is still required to store a subset of data locally in order to provide a service. For example, in the case of an image, the web application may request and store meta-data relating to the size of the image to assist in rendering pages during the presentation stage. This also opens up the possibility of web applications temporarily caching data, and is addressed in future research using cache-coherency protocols.

C. Presentation

The final stage in the design, depicted in figure 6, is the presentation stage. Here we define how the data is transparently presented to end users. Again the browser extension plays a key role. In this instance the extension operates as a 3rd party does not have any direct relationship to the DDS of the rendered data. For example, an end user may access a web application and request to view an image collage built from images stored in multiple DDSs owned by multiple data owners.

In this instance the web application, instead of returning raw data as in the classic Web 2.0 case, will return a *DDS-Present-DataRequest* containing security information exchanged between the web application and DDS during the initial authentication request. This security information is protected using PKI to ensure that it cannot be abused to falsify links between a DDS and unauthorised web applications and clients. The trust relationship enforced in this case is between the web application and the DDS, hence the DDS itself does not need to be aware of all the end users who can render the data linked to a specific web application. The *DDS-Present-DataRequest* message triggers a handoff of the user from the web application to the DDS, allowing the browser extension to request and render the data directly from the DDS, under the web application's instruction.

Under the DDS model, clients are required to perform additional processing to pass-through and cater for the DataRequest messages. Actual data transfer and rendering functions remain largely unaffected other than the fact that the web browser, on average, would be compiling single pages from multiple data sources. These data sources would be a combination of static data from the web application and dynamic data sourced from one or more DDS systems. Web page rendering engines in modern web browsers already support rendering a single page from multiple components so actual page rendering will appear identical to the end user when compared with current solutions.

IV. Prototype and Example Runtime

A proof-of-concept prototype was developed to demonstrate all of the components described in section III. The prototype comprises the following:

- A skeleton distributed data service implemented in Java and utilising the Amazon S3 service for data storage.
- A proof-of-concept DDS-enabled web application, implemented in PHP, which is capable of subscribing to a DDS and linking image and postal address data.
- A Mozilla Firefox web browser extension implemented in Javascript to provide the data owner and end user experience.

Multiple programming languages were selected for the prototype to demonstrate the programming language-agnostic nature of the DDS API.

The domain of the prototype is an in-house group address-book application with which users can create a profile and upload their office address and a profile photo. The address and photo data are stored in the data owners DDS. The following is an example runtime flow from the prototype application. It describes a user linking some data and a second user in turn rendering that data.

- User A (Bob) accesses the website for his DDS of choice and begins the registration process
- DDS(Bob) sends an attachment request message (through the HTML response) that is detected by his web browser extension
  - *DDS:DDS-StorageService-AttachRequest() to Extension(Bob)*
- Extension(Bob) requests Bob's approval to attach to the DDS and sends a successful response message
  - *Extension(Bob):DDS-StorageService-AttachResponse(SUCCESS) to DDS(Bob)*

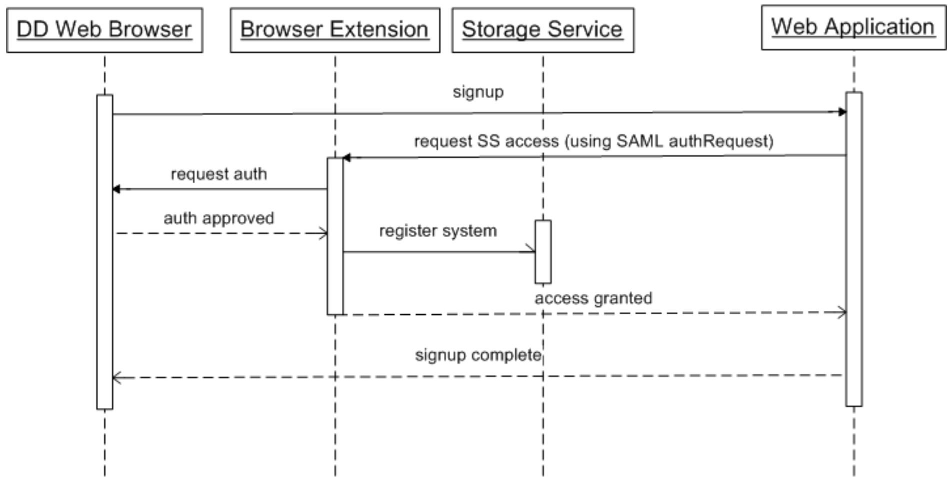

Figure 4. Web Application Registration

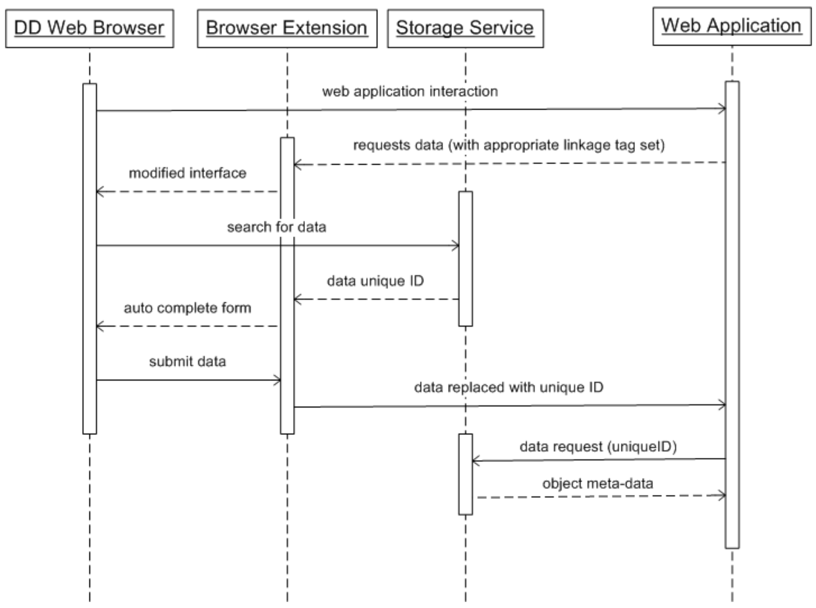

Figure 5. Data Linkage

- Bob then continues to interact with the website presented by the DDS to upload his office address and profile image data
- Bob now accesses the website for the address book application (WebApp) and begins the registration process
- The Web Application sends an attach request to the browser extension in Bob's browser. The request is signed with the web application's private key and includes a copy of the web application's public key for verification.
  - *WebApp:DDS-Application-Subscribe-AuthRequest(publickey(WebApp), WebApp) to Extension(Bob)*
- The browser extension requests Bob's authorisation to allow that web application to subscribe to data within his DDS and sends back a response. The browser also forwards the request on the DDS so the DDS can locally register the request.
  - *Extension(Bob):DDS-Application-Subscribe-AuthResponse(SUCCESS) to WebApp*
- The web application then allows Bob to upload an image. Attached to the standard INPUT HTML element an additional DDS-Enabled="true" attribute is included. This instructs Extension(Bob) to render that input element as a DDS data-lookup field.
- Bob selects his profile image from the pop-up DDS interface and the browser extension provides the unique ID of the data ID(image) back to the web application

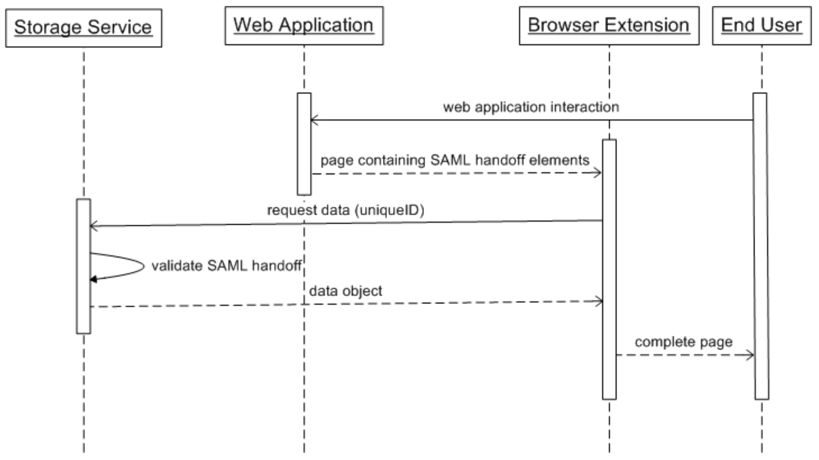

Figure 6. Data Presentation

for storage.

- User B (Alice) now accesses the website for the address book application and asks to view Bob's profile.
- The web application inserts a data request message into the HTML response that is received by Extension(Alice).
  - *WebApp:DDS-Present-DataRequest(publickey(WebApp), ID(image)) to Extension(Alice)*
- Alice's web browser extension then establishes a direct connection to Bob's DDS using the system code provided in ID(image) and forwards on the data request.
- DDS(Bob) authenticates the request by validating the signature of the message using the publickey(WebApp) established during the authentication request stage.
- DDS(Bob) then returns the image for rendering by Extension(Alice).

V. System Evaluation

A. Functional Requirements

The model presented in this paper sets out to solve multiple issues stemming from the traditional monolithic storage approach used by web applications on the Internet.

Distributing data element storage greatly reduces the resource requirements of web applications. Storage requirements will decrease to only those needed to store meta data on the web application itself rather than the user-generated content. Bandwidth requirements for the web application will drop as the application will only be returning basic HTML, CSS, script and meta-content such as logos and branding. All user-generated content will be directly transferred to the end-user from the related DDS systems. Lastly, the web application provider will no longer be required to meet varying privacy legislation requirements as they will not be directly storing any user's personal data. This requirement is instead offloaded to the DDS providers, which can operate in the same geographical region (and hence be subject to the same legislature) as the data owner.

From the end-user perspective, the model addresses the data freshness issue by replacing the N multiple copies of a data element with N links that point to a single instance of the data stored in the data owner's DDS. These N links are abstracted using DNS technology to ensure that a user can migrate from one DDS to another without invalidating the links. This removes the danger of a system accessing obsolete versions of data by creating a single version of the truth for every data element in the system.

The model also addresses the data duplication-created storage wastage issue. This is true provided the number of bytes used to store a link is less than the number of bytes used to store the actual data. With this assumption we achieve a reduction of the storage requirements in a single system from (M * objects) to (N * objects) where N is the size of a link and M is the average size of stored objects. A web application would only be required to store the [system] component of the link once per user.

Most importantly, the issue of data ownership is also addressed. Use of owners' data was previously dictated by web applications, and was typically enforced by end-user licensing 'agreements'. If a user wished to use a particular web application they had no choice but to accept the EULA. With the described DDS model, the user has more freedom. It is safe to assume that the DDS itself may also enforce a EULA on the end-user, but in this situation the user has the buying power to procure services from another DDS provider that requests a less restrictive license.

Security of the DSS system is provided in multiple layers. All communication between the data owner and the DSS can be protected using such existing technologies as SSL. Basic authentication would suffice when the transport layer is protected. The link established by the data owner with the web application forms the basis of an authentication token which is then used to authenticate end-users through the web application into the data owners DSS. This handoff is protected using the SAML handoff framework. All security assertions as signed with the DSS validating the signature as belonging to the data owner. This allows the DSS to ensure that any request for data coming from an end-user, through a specific web application, has been authorised by the data-owner.

B. Comparative Evaluation

While the authors could not find any other model specifically targeting the core issues of this research, there are systems that are similar in nature to the DDS.

1) CMS: Parallels can be drawn between the DDS and Content Management Systems [7]. The DDS can be viewed as a personal CMS that allows its content to be seamlessly embedded into 3rd party web applications. The DDS provides distributed storage of user content for web applications that previously relied on monolithic storage repositories.

Current CMS solutions do not scale to the level required to implement an Internet-wide distributed storage solution due to their own reliance on monolithic storage structures.

2) CDN: Content Delivery Networks [8] are distributed storage networks that allow companies to host data objects on 3rd party networks. This allows them to take advantage of geo-location based load balancing and link peering to achieve reduced bandwidth costs. The typical CDN solution is similar to the delivery paradigm in the DDS model except that CDNs do not currently provide a seamless way for data owners to push content into the network and have that content transparently accessed by authorised web applications. CDNs are static in nature, and do not scale to the dynamic features that the DDS model provides.

3) Cloud SSP: The presented design ties directly into the realms of Cloud Computing [9], Service-Orientated architectures [10] and SAAS (Software-as-a-service) [11]. In a sense, a DDS can be seen as a SSP (Storage Service Provider) in a Storage-as-a-service [12] cloud component that allows other web applications to publish and subscribe to data within the cloud. The DDS model described in this paper, however, provides the necessary additional access and presentation layers on-top of the storage to ensure that the user experience is seamless.

Cloud computing can play an important part in the design and hosting of the DDS storage system itself. As the DDS API does not explicitly define the internal design of the DDS, the vendor is free to, for example, use Cloud Computing technologies, this providing a DDS solution that benefits from the dynamic scalability and per-usage business models that the Cloud provides.

C. Performance

The paradigm shift described in this paper dictates a movement of data storage away from classic monolithic storage, towards a distributed network of data storage services. As such, performance has been analysed to identify overheads introduced by the additional access and presentation complexity. While a small constant overhead was identified due to the web application-to-DDS handoff requirements, performance increases were also identified in the following areas:

- Speed improvements under high-load situations due to the reduced data transfer requirements of web applications. Instead of the web application being responsible for provision of all the displayed data, the data transfer requirements are shared between the web application and the various linked distributed data services. This has the potential to reduce the network load of web applications.
- Client geo-locality can be utilised when users access data that is geographical in nature and when the required distributed data services are located closer to the client than to the web application. For example, a user browsing images of their friends on a social networking site would experience improved performance if the DDSs for their friends were less network hops away than the social networking web application itself.
- Speed improvements were identified in cases in which a single webpage is built of multiple separately loaded elements, a popular model in systems that rely heavily on user-generated content. In the general Web 2.0 case, browser pipelining restrictions limit the number of simultaneous network requests to any server. By distributing data storage the impact of these restrictions is reduced. Figure 7 shows the performance improvements for the case in which a single page is built from multiple data elements, each separately sourced. The experiment was performed with a pipelining restriction of four simultaneous connections per server. The results show a marked improvement using the DDS system.

Performance of the DDS system can be adversely affected by poor connectivity and bandwidth to specific DSS nodes. The open market for storage services will assist in driving competition between storage providers to reduce this risk. Obviously there are upfront costs involved in the integrating web applications with the DDS API, but these are offset by reduction in ongoing costs bandwidth and storage costs.

D. Backwards Compatibility

In respect to backwards compatibility, the system supports the ability for non-enhanced web browsers to retain access to DSS-enhanced websites. Such access would be facilitated using a 'proxy' model where another system resolved and presented the data from the DSS systems in a transparent manner to the non-enhanced web browser.

These proxies could either reside as part of the web application or as an intermediate service provided by the end-users Internet Service Provider. While involving an additional level of redirection reduces some of the benefits

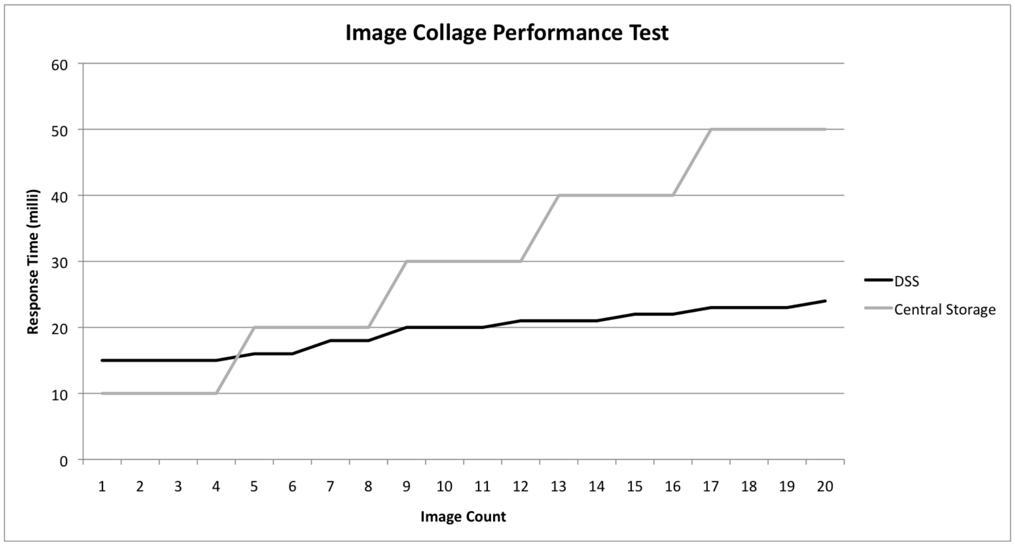

Figure 7. Performance Results

provided for the end-user by the DSS model, it retains the complete set of benefits for the data owner and web application owner.

VI. Scalability

The DDS system scales exceptionally well due to the decentralised nature of the data storage. Each user is free to choose their own DDS host(s). As the user base grows, the number of DDSs linked in a web application also grows. Each unique DDS can execute within a cloud computing environment, hence internal scalability is also supported in the situation where large numbers of data owners choose to use the same DDS (for example, all users of a particular University may choose to use a University-hosted DDS solution).

The DDS solution also scales into the corporate space where each corporate entity could host their own DDS. This would allow the employees of a company to share data internally, as well as externally through restricted publish/subscribe functions. The openness of the DDS API allows corporate entities to protect their data by controlling which web applications subscribe to specific pieces of data.

From an end-user perspective, the DSS system scales in the same fashion as a traditional client/server model. The transparent nature in the way the DSS browser extension provides visibility of DSS-stored information ensures that the end-users experience remains unaltered. From a connection viewpoint, the constant overhead described in the performance review above has no effect on the solutions ability to scale when compared to traditional approaches.

VII. Conclusion

This paper addressed three concerns resulting from the growing popularity of Web 2.0 applications by formally defining a new paradigm for the distributed storage of data on the Internet. The standard for web applications has evolved, from static pages comprising a limited number of elements to complex pages rendered from a large numbers of elements. Web 2.0 has seen a trend towards bandwidth intensive elements originally generated by end-users. As the user take-up of Web 2.0 applications continues, it is sensible to adopt a distributed approach that parallels the way content is originally generated.

Problems caused by the usage of monolithic data storage features have been mitigated by adopting a distributed storage approach. Moving from monolithic to distributed structures is a proven technique for sharing load that has been used extensively in other areas such as Cloud Computing [9] and Transaction Management [13].

The key issue of data ownership is addressed for end-users by ensuring that storage is the responsibility of distributed data service(s) directly engaged by them. DDS providers are liable to data owners, not to web applications, and hence data owners have control over use of their data. Data ownership is clear-cut because owners are responsible for both storage of, and access to, the data.

Data freshness is addressed using a publish/subscribe model and an enhanced SAML-based handoff model for data presentation. The data rendered in web pages is always the freshest version because it is sourced directly from the data owner's DDS. Data duplication is also addressed by removing the need for data to be stored by web applications. Appropriate web application registration and linking reduces the number of copies of any piece of data to a single instance stored in the DDS.

Current modelling and experiments show that overall system performance is comparable to the existing Web

2.0 paradigm in the general case, with minor constant overhead caused by the handoff procedures. When a web application renders a page containing multiple data elements from multiple DDS repositories, we observe a performance improvement compared to existing technology due to the bypassing of web browser pipelining restrictions.

Comparisons made against similar systems show that the new paradigm can greatly increase the quality and protection of data in a Web 2.0 space. For the DDS model to become widely utilised the DDS API will need to be adopted as a standard.

References

[1] T. O'Reilly. (2005) What is web 2.0. [Online]. Available: http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html. (cited June 2010)

[2] M. Wallis, F. Henskens, and M. Hannaford, "Publish/subscribe model for personal data on the internet," in *6th International Conference on Web Information Systems and Technologies (WEBIST-2010).* INSTICC, April 2010.

[3] AFP. (2009) About-facebook: backflip on data ownership changes. [Online]. Available: http://www.smh.com.au/articles/2009/02/19/1234632933247.html. (cited June 2010)

[4] F. Henskens, "Web service transaction management," *International Conference on Software and Data Technologies (ICSOFT)*, July 2007.

[5] K. Feldt, *Programming Firefox: Building Rich Internet Applications with XUL (Paperback).* O'Reilly Media, Inc, April 2007.

[6] OASIS, "Security assertion markup language (saml) v2.0 technical overview," Working Group, Tech. Rep., 2007.

[7] A. Mauthe and P. Thomas, *Professional Content Management Systems: Handling Digital Media Assets*. Wiley, 2004.

[8] M. Hofmann, *Content Networking: Architecture, Protocols and Practice*. Morgan Kaufmann Publishers, 2005.

[9] G. Boss, P. Malladi, D. Quan, L. Legregni, and H. Hall. (2007, October) Cloud computing. [Online]. Available: http://download.boulder.ibm.com/ibmdl/pub/software/dw/wes/hipods/Cloud_computing_wp_final_8Oct.pdf. (cited June 2010)

[10] M. Bell, *Introduction to Service-Oriented Modeling, Service-Oriented Modeling: Service Analysis, Design, and Architecture*. Wiley and Sons, 2008.

[11] K. Bennett, P. Layzell, D. Budgen, P. Brereton, L. Macaulay, and M. Munro, "Service-based software: the future for flexible software," in *Seventh Asia-Pacific Software Engineering Conference (APSEC'00)*, vol. 17th, 2000, p. 214.

[12] J. Foley, "How to get started with storage-as-a-service," *InformationWeek Business Technology Network*, 2009.

[13] F. A. Henskens and M. G. Ashton, "Graph-based optimistic transaction management," *Journal of Object Technology*, vol. 6, no. 6, pp. 131–148, July/August 2007.