Seadrive

Peter Haro[1,2] and Otto Anshus[2]
[1]SINTEF ICT, Trondheim, Norway
[2]Department of Computer Science, University of Tromsø, Norway

Abstract

Seadrive is a novel file synchronizations framework and file hosting service for offshore naval fleets. Offshore vessels outside the range of cellular networks employ volatile satellite-based networks unsuited for file synchronization. Seadrive aspires to provide naval fleets with usable file hosting services over low bandwidth, high latency network links with high loss rate and large variability in the overall network stability. We provide the rationale for Seadrive; present the architecture, and design of the framework. We present an initial evaluation of the correctness and performance, preliminary results shows that Seadrive outperforms existing system in the test environment.

Introduction

Modern naval- and fishing fleets utilize a multitude of information systems from different sources when planning and executing offshore operations. These vessels are equipped with several sensors and instruments, which provide a constant stream of information regarding various on-board systems, of which some are readily available to the crew and actively employed during an operation. The governing bodies of large naval-, fishing-, and oil- fleets requires information to flow from their management system(s) to their fleet in a robust manner, likewise the fleets have information required by the governing entity.

The information flow in today's systems is primarily based on using e-mail. However, end-users report these systems as unsatisfactory for the following reasons.

- Important information is lost in the copious amount of other e-mails
- Documents arriving during shift A are often not read by shift B
- The systems are slow
- The files sent through these systems are constrained not only by file-type, but also their size

When important information is lost, the results can vary from small trifles to disasters, such as overfishing, not following updated safety regulations or monetary losses. Therefore, entities that manage large fleets have experimented using cloud-based file-synchronization frameworks to deliver data to the end-users. Although there exists a myriad of file synchronizations frameworks and file hosting services such as "Dropbox" [2], "Rsync" [3],, "Google drive" and many more, the systems do not function correctly for ships connected to the internet through satellite based network-links. The connections are plagued with high packet loss and frequent dropped connections, which causes several problems for the file-synchronization services. None of the aforementioned services managed to successfully synchronize a file to land, in a preliminary test on an offshore vessel. We determined the primary cause of this to be the frequent allocation of new IP addresses, due to dropped connections and restarting the synchronization process during abrupt changes.

To provide a robust framework to dispatch, read and update files, in order to accurately disseminate the required information to end-users, we propose a file-synchronization application and framework: Seadrive. The primary objective of the Seadrive project is to establish a robust file synchronization framework to disseminate data in a many to one, and one to many relationships. In order to achieve the objective, the system must support stable transmission over unstable network connections, allow

This paper was presented at the NIK 2016 conference. For more information see http://www.nik.no/

for retransmissions of a subset of a file, and be able to rebuild the file regardless of type after transmission.

Seadrive architecture and design

Seadrive provides a framework that consists of several different components, where each component encapsulates a particular functionality. These components are interconnected to work in unison conducive to create the client- and server side functionalities. In order to complete the primary objective of Seadrive, the architecture must accommodate for physical constrains such as hardware, network bandwidth, loss ratio, and network topology. A detailed description of the current system can be found in [1].

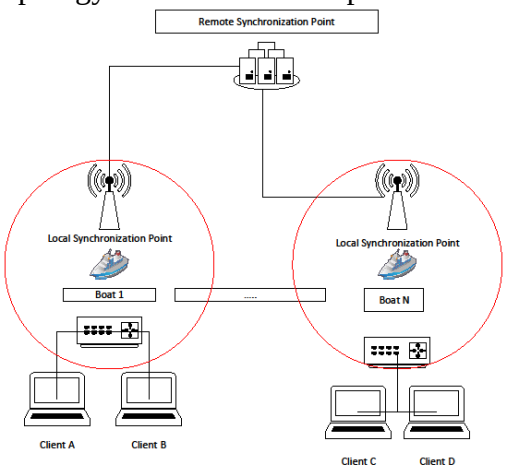

Figure 1 Architecture of Seadrive. Clients are reciprocally synchronized within the Local Synchronization Point, and is continuously synchronizing with the Remote Synchronization Point whenever possible.

For this short paper, we delimit ourselves to focus on the components outlined in Figure 1, emphasizing the local- and remote transport protocol.

The quintessential aspect of the framework is the division of intra- and inter communication. The intra communication between clients and the local synchronization point (LSP) uses a lightweight transport protocol, designed for speed and simplicity. The remote transport protocol between local and remote synchronization points (RSP) is designed for robustness and to use minimal bandwidth.

Local Synchronization Protocol

The local synchronization protocol is the process of synchronizing all the data in a given repository within a vessel. It is the primary method of communication between each client interconnected in the *LSP*. Its responsibilities include managing uploads and downloads for all entities interconnected in the *LSP*.

The local synchronization protocol between *LSP*s and its clients is a two-step process. A set of metadata concerning the changed file(s) is transmitted from sender to receiver in order to validate that the files have actually changed. The receiver responds by requesting changes, if there are any in one of two methods. The simplest methodology is to request the entire file in order to rebuild it from scratch. The other is to request the missing chunks in order to patch the file. For *LSP*s, once it has received a new file, the changes are disseminated to all clients through a broadcast mechanism. To ensure that the data has been transferred completely, the receiving entity compares the checksum of the rebuilt file with the checksum(s) received in the negotiation.

Remote synchronization Protocol

Remote synchronization is the process of synchronizing files between the *LSP* to the *RSP*, and vice versa. This synchronization is the critical mechanism that makes offshore-based file synchronization possible. Like the clients, the *LSP* monitors changes within itself, which creates a changed-set, consisting of dirty files and a given data-deduplication and synchronization mechanism. Once the changed-set is prepared and ready for transportation, the *LSP* initiate contact with the *RSP* to negotiate the transmission.

Unlike the Local Synchronization Protocol, the remote protocol utilizes an explicit algorithm for the transmission of data in several well defined steps. The remote synchronization protocol is designed to be a stateful algorithm, in which every operation must correspond to a pre-determined state. This means data sent out-of-order between expected states must retransmit from a previous state known to both the sender and recipient.

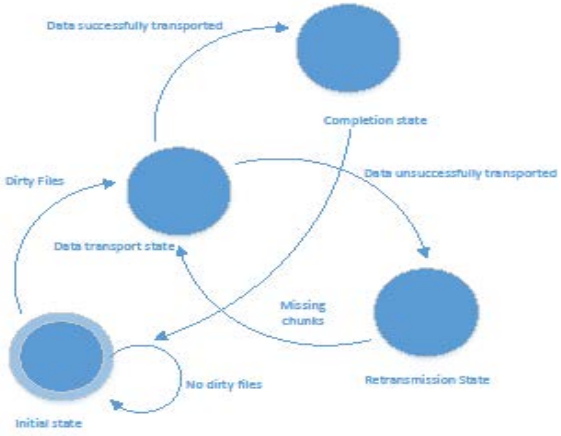

Figure 2 State diagram of the sender in the remote transport protocol.

The remote transport protocol utilize five states in order to reliably transfer data to remote entities. The behavior of the system depending on its current state can be found in [1]. These states allow the system to support retransmission of data either for dropped packets or for connections. The states allow the land-based entity denoted as RSP in Figure 1 to maintain a serializable in-memory cache of all registered client, most often LSPs. This allows for rapid retransmission of data as clients can continue where the previous session ended.

Initial evaluation and results

Despite the fact that the implementation of Seadrive is not yet feature complete, it is possible to evaluate the current implementation in respect to the taxonomy of file synchronizers [4] such as data exchanged between synchronizing devices, computations, network size, robustness and memory.

Experimental Design

We created micro-benchmarks in order to evaluate the taxonomy of various file-synchronization schemes. We have conducted the benchmarks on a reference implementation Octodiff[1], which is an optimized version of Rdiff, implemented in C#.net for usage in Octopus Deploy. Furthermore, we utilize our own version of librsync, based on the .net port publicly available on Github[2], as the original .net

[1] https://octopus.com

[2] https://github.com/braddodson/librsync.net

implementation was too slow for our needs. Finally, we compare these results to the Seadrive Binary patch utility based on delta-differental.

The experiments used a single machine implementation of the applications with 64-bit Microsoft Windows 7 OS and were run on a DELL Latitude E7440 with the following specifications: Intel Core i7-4600U @ 2.10 GHz with four cores, Intel Graphics, 16GB ram @ 2100 GHz, and a LITONIT-LMT-256 SCSI disk.

Results

The first micro-benchmark measures the average size in bytes the methodology needs to send over the network in order to synchronize the file(s). In this experiment, the Seadrive binary differential method achieves the highest range of compression averaging out at 94%, compared to the runner up Octodiff, at 90%. A complete overview of the datasets, metrics and results can be found in table 8.1 in [1].

The latency-measurements found in [1] shows that sliding window protocols are significantly faster than an algorithm based on delta-differentials with suffix array sorting. They run in an almost negligible time in our use case, with Octodiff as the fastest method clocking in at ~8.7 seconds. We observe a significant overhead of running the Seadrive binary diff protocol, as it is ~17 times slower than Octodiff at producing the patch-files and uses on average 154 seconds to produce the patch.

Combining the results of the aforementioned benchmarks, we can see that in order to select algorithms for synchronizing files one must look at the total time required to patch the files on the actual networks. These networks provide a maximum bandwidth available to the application at 8kb/s, 4kb/s and 512 bits/s, where the maximum throughput is dependent on the data plan. For the following results in table 1, we measure the time in hours required to transfer the average data required to successfully synchronize the files. We ignore the costs of multiple transmission required by the Rsync algorithms, as they are negligible in the time windows presented.

	LibRsync	Octodiff	Seadrive binary-diff
8 kb/s	1.66 H	1.55 H	0.96 H
4 kb/s	3.32 H	3.11 H	1.93 H
512 bits/s	25.99 H	24.35 H	15.05 H

Table 1 Shows the time to transfer the delta-files over various dataplans in hours

We note that the time saved by the Seadrive-binary diff methodology is significant, by a 1.5 order of magnitude. This causes the time spent to generate the binary diffs insignificant; furthermore, it saves a huge fiscal cost, as each byte sent is equivalent to monetary values. By the virtue of these results, the binary diffs are clearly the optimal method to save both time and monetary values.

Conclusion

This short paper has introduced and described Seadrive, a new file synchronization framework, and we have placed special emphasis on the remote transport protocol. Our contribution delivers a new methodology to remotely synchronize files.

References

[1]. Peter Halland Haro. Seadrive - Remote file synchronization for offshore fleets Master's thesis, Dept. of Computer Science, University of Tromsø. April 2016 http://munin.uit.no/handle/10037/9373

[2]. Idilio Drago, et.al. *Inside dropbox: understanding personal cloud* storage services. *In Proceedings of the 2012 ACM conference on Internet measurement conference, pages 481–494. ACM, 2012.*

[3]. Andrew Tridgell. *Efficient algorithms for sorting and synchronization*. Australian National University Canberra, 1999.

[4]. Sachin Agarwal, David Starobinski, and Ari Trachtenberg. On the scalability
of data synchronization protocols for pdas and mobile devices. *Network, IEEE, 16(4):22–28, 2002.*