# Towards Dispersed Cloud Computing

Josef Spillner, Alexander Schill
Faculty of Computer Science
Technische Universität Dresden
01062 Dresden, Germany
Email: {josef.spillner,alexander.schill}@tu-dresden.de

***Abstract*—The growing number of locally and globally connected devices and infrastructure services is symptomatic for increasing user requirements on convenient, comfortable, fast and cheap remote computing and access to data. Distributed computing techniques in combination with on-demand Clouds help matching the requirements and the offered resources. This paper evaluates existing distributed computing models according to their specific code, data and resource handling and defines a specific one, called dispersed computing, by consolidating existing research on dispersed data transmission, data storage and code execution. It furthermore contributes a mapping of the model to typical Cloud environments. Complementary, we share our experience with dispersed computing implementations.**

## I. Distributed Computing Models Background

Computing models serve as foundation for the programming of software. Historically, computing has evolved from the single-core architecture, the addition of co-processors, parallel computation on multi-core processors, distributed parallel computation, hybrid models with heterogenous processors and finally service-orientation and Clouds which offer more dynamic environments into which the computing tasks can be offloaded on demand. Along with the evolving models, appropriate programming support has co-evolved through libraries, toolkits, new languages and infrastructure services [1].

In the following three subsections, we briefly present distributed computing techniques applied to the commonly used infrastructure resources: networks for communication, storage media and processing nodes. Although there are more fine-grained infrastructure service types in use [2], most of them are subsumed by these three. We do not claim complete coverage of all distributed computing fields; instead, we provide sufficient background information for differentiating the dispersed computing model from other techniques.

### A. Distributed Communication

The Internet is a distributed communication system which offers routing, load balancing and network-embedded services (IP Multimedia Subsystem (IMS), Information-Centric Networking (ICN), Carrier Clouds). Based on these primitives, the application-level division of client-server traffic flows is performed through peer-to-peer applications, multipath connections and anonymisation techniques [3].

### B. Distributed Storage

Storing data on more than just one drive or other medium is possible through several techniques. On the block level, JBOD, RAID and further combinations of local and remote disks distribute data efficiently and, depending on the configuration, safeguard against performance bottlenecks and failures. Distributed and global file systems, on the other hand, bring awareness of the distributed states into the higher layers of the storage stack. More recently, combined storage service systems have become popular. Researchers are also looking into distributed object stores, multiple distributed hash tables, spatial indexing [4] and other improvements to the management and organisation of distributed storage.

### C. Distributed Computation

The field of distributed computing is huge. The parallel flavour includes decentralised volunteer computing, partially driven by the aim to efficiently use idle resources, but also centrally controlled nodes such as mobile agents and botnets. In contrast, the federation flavour is often seen in practice due to the trade-off between control and convenience. Many Internet services, including the web, are large federated systems with different degrees of compute or processing services. With the emerging trend of processing large volumes of data, the Map-Reduce programming model which parallelises compute steps by mapping input to output before performing a central reduction step has become popular. There are several implementations, such as the popular Hadoop, extensions (e.g. for relational data structures) and optimisations including YARN and SailFish [5]. Some approaches also focus on fault tolerance and real-time processing, for instance StreamMine3G [6] which incorporates complex event processing and event stream processing functionality. Most of these models assume unrestricted access to all data required for the computation.

## II. Dispersed Cloud Computing Model

Information dispersal algorithms divide data into $n$ fragments, each of the length $1/m$, so that any $m$ of them are sufficient to unify the original data [7]. They are sporadically implemented in storage and communication networks [8]. With a growing number of devices, services and data sources, information dispersal will become more commonplace to protect against quality issues in distributed systems.

We define *Dispersed Computing* to be a specific model of distributed computing in which data is not divided along the logical unit boundaries, i.e. variables or structures, but instead the units are divided into subunits in a way that there is either exactly one or there are multiple inverse divisions

to reconstruct the original units. Dispersed Computing also implies a set of techniques following the model to perform the division and reconstruction. Furthermore, we define *Dispersed Cloud Computing* as the application of Dispersed Computing techniques to Cloud environments, primarily on the infrastructure level with compute, storage and network resources.

In our notion, an *unreliable resource* is one which can either fail, or be forcefully deactivated, or be intercepted, or become slow, or in any other way offers a quality which is lower than the expectation of the user. Dispersion implies that multiple similar resources are used in parallel, at least one for each subunit, in order to protect agains the unreliability. It also implies a-priori treatment of data at the user's discretion..

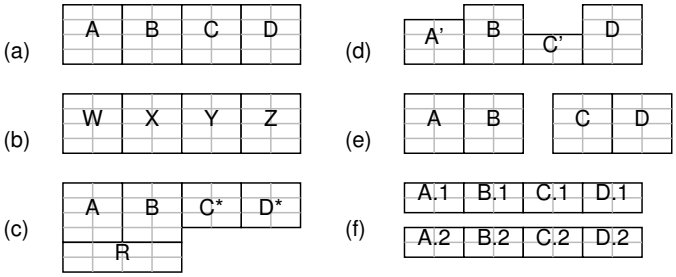


Fig. 1. Data treatment possibilities on the bit level

Fig. 1 compares the treatment of four data units under the Dispersed Computing model to other forms of modification. Each grey rectangle represents one bit. The figure shows the original data (a), encrypted data (b), redundantly encoded or compressed data (c), meaningful adapted data (d), chunked data (e) and dispersed data (f). Both for (e) and (f) the term splitting can be used. Indeed, both are often combined in order to disperse large volumes of data or streaming data.

The following subsections describe the specifics when using this model of distributed computing and describe a mapping to common infrastructure resources in the Cloud.

### A. Dispersed Communication

In dispersed communication, a client node (called sender) splits the data to on-demand network links and a server node (called receiver) reconstructs it.

While conventional channel balancers select one transmission channel per connection request from the application, a dispersing channel balancer selects a set of transmission channels per request. Some of these may be redundant, which implies a continued operation as long as less or equal the number of permissible channels fail. The selection is based on multiple factors, among them order (for round-robin selection), randomness, load (as reported by real-time service monitoring), content of the data and context from the environment.

Dispersed communication can improve some specific transmission metrics, but never all of them at once. The metrics are the parallel throughput, the reliability and safety, the security against eavesdropping and the resilience against resource changes. One possible application would be mix cascades which conceal the purpose and origin of traffic flows. Another one would be channel bonding among several network interfaces, for instance a wired and a wireless adapter.

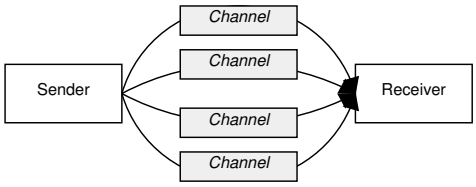


Fig. 2. Dispersed communication

Fig. 2 represents a dispersed communication scheme. The channel itself is considered as unreliable resource here, while sender and receiver are considered stable and trustworthy.

*Dispersed Communication Characteristics:* Table I lists major characteristics of dispersed communication configurations. Of interest is the correspondence between split and join operations and the resulting theoretic traffic overhead. In the case of coding, a selected amount of redundancy is introduced which depends on the encoding scheme, e.g. Cauchy-Reed-Solomon or AONT-RS for more secure secret sharing at the cost of full replication. In practice, there may be additional overhead due to padding, as well as overhead in other resources, such as the CPU for performing the split and join operations.

TABLE I
DISPERSED COMMUNICATION CHARACTERISTICS

| Mode | Split | Join | Overhead |
|---|---|---|---|
| Parallel | Chunking | Concatenation | 0% |
| Dispersed | Splitting | Concatenation | 0% |
| Dispersed | Coding | Selection, Recod. | 0%-100% |
| Replicated | Replication | Selection | 100% |

*Example for Dispersed Communication:* Two numbers, say 71 and 19, are to be sent across a network with two channels. With per-unit splits, each number would be sent on one of the channels. With dispersion, both numbers are split as shown in Table II. At the receiving end, both transmitted flows are joined again and the original data is retrieved. This calls for a synchronisation which can either be done through the network stack (i.e. all sockets need to have a minimum amount of readable data) or through the application, which is more complex but more scalable due to the usually limited network buffers in operating systems.

TABLE II
TRANSMISSION OF TWO NUMBERS WITHOUT REDUNDANCY

| Number (decimal) | Number (binary) | Fragments |
|---|---|---|
| 71 | 01000111 | $a_1$: 0100; $b_1$: 0111 |
| 19 | 00010011 | $a_2$: 0001; $b_2$: 0011 |
| Transmitted data is $a$: 01000001; $b$: 01110011 | | |

### B. Dispersed Storage

In dispersed storage, a client node splits the data when storing data onto a server (called target) and reconstructs

it when retrieving it. This offers similar characteristics to dispersed communication. One advantage is that it works purely client-side so that it applies to many real-world scenarios in which the server side is not fully under the user's control. Hence, dispersed storage is often applied to long-lasting, reliable and secure storage of personal data in the Cloud. These characteristics in part depend on the technical and organisational independence of the server nodes.

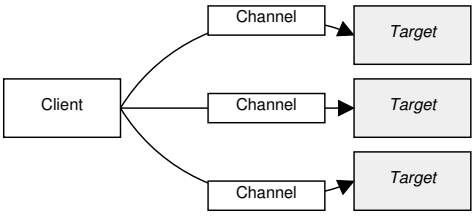


Fig. 3. Dispersed storage

Fig. 3 represents the basic dispersed storage scheme. In combination with dispersed communication and general data flow architectures, recent research work calls for user-defined and user-controlled storage flows.

*Example for Dispersed Storage:* There are practically no differences to the previous example on dispersed communication. However, the same example can be given again with additional redundancy. The chosen redundancy scheme is a simple XOR which is suitable for understanding although rather inflexible in practice. Assuming three storage targets are available, the dispersion are split as shown in Table III.

TABLE III
STORAGE OF TWO NUMBERS WITH 50% REDUNDANCY

| Number (decimal) | Number (binary) | Fragments |
|---|---|---|
| 71 | 01000111 | $a_1$: 0100; $b_1$: 0111 |
| 19 | 00010011 | $a_2$: 0001; $b_2$: 0011 |
| $a_n$ XOR $b_n$ | | $c_1$: 0011; $c_2$: 0010 |
| Stored data is $a$: 01000001; $b$: 01110011; $c$: 00110010 | | |

### C. Dispersed Computation

In dispersed computation, a client node splits the data to be operated on. Server nodes (called processors) perform their operations on partial data and return partial results. The client node then reconstructs the full result. As opposed to working on chunks of data, dispersed computation either needs to restrict or to modify the semantics of operations depending on the dispersion. This is similar to homomorphic encryption operations which work on encrypted data without distribution. Dispersed computation is clearly an open research topic as it presents additional challenges.

A suitable taxonomy of dispersed computation is driven by the types of data to be worked on. These encompass mathematical formulas for numeric types, string operations, and more complex ones such as split objects in which not all methods or member variables are available on each node.

Fig. 4 represents the scheme behind dispersed computation. Similar to the Map-Reduce model, the client's task is to recombine the partial results from the distributed processors. This makes the client likely the bottleneck. However, as opposed to large-scale distributed chunks, dispersion is frequently used with rather modest numbers of fragments due to the small size of primitive data types. Therefore, this restriction is clearly less severe. On the other hand, the overall performance of the computation will be negatively impacted by the communication and coordination overhead.

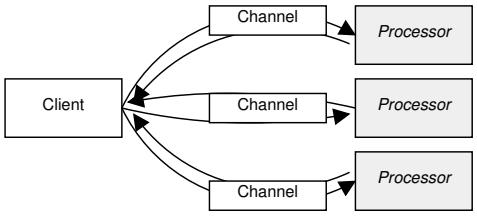


Fig. 4. Dispersed computation

*Example for Dispersed Computation:* Table IV gives a simple example of a dispersed addition operation. A carry bit may occur in the output of each node which needs to be taken into account when designing the data type sizes. Already for multiplication, a two-node distribution will not work in all cases anymore.

TABLE IV
ADDITION OF TWO NUMBERS WITHOUT REDUNDANCY

| Number (decimal) | Number (binary) | Fragments |
|---|---|---|
| 71 | 01000111 | $a_1$: 0100; $b_1$: 0111 |
| 19 | 00010011 | $a_2$: 0001; $b_2$: 0011 |
| 71+19=90 | 01011010 | |
| Partial results are $a_1$+$a_2$: 0101; $b_1$+$b_2$: 1010 | | |
| Computed result is ($a_1$+$a_2$ « 4) + $b_1$+$b_2$: 01011010 $\hat{=}$ 90 | | |

### D. Combined Use with Infrastructure Services

While dispersed computation is subject to heavy performance penalties, the combined use of dispersed storage and computation offsets this by exploiting the crucial issue of data locality. For instance, Cloud providers often offer both compute and attached local storage services. This allows for distributed calculations over redundantly dispersed data. Even in the case of service failures, the results can still be safely retrieved. Conventional fault tolerance mechanisms which rely on checkpointing and restarts in operations over chunked data can only achieve the same level of reliability with a higher code complexity and with less autonomy over the data.

Fig. 5 visualises the combined use of dispersed computation schemes in a typical cross-provider Cloud application scenario.

## III. SOFTWARE IMPLEMENTATIONS

Dispersed computing is not yet a dominant paradigm even in Cloud environments where most of its advantages apply. There are however a number of research prototypes and production tools available. In this section, we give an overview with initial evaluation but without comparison or rating of the software.

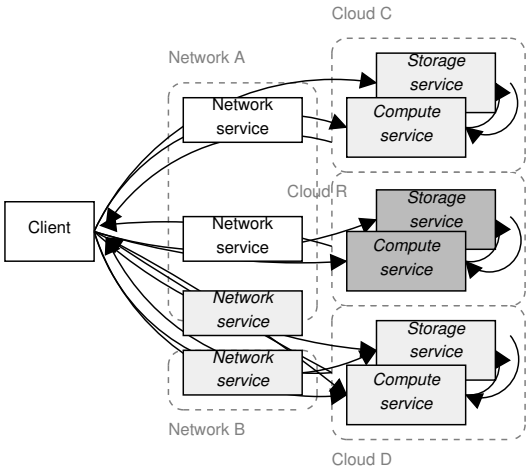


Fig. 5. Combination of partially dispersed communication (A/B) with 0% redundancy as well as fully dispersed storage and computation (C/D) with 50% redundancy (R) in a typical Cloud environment

### A. Dispersed Communication Software

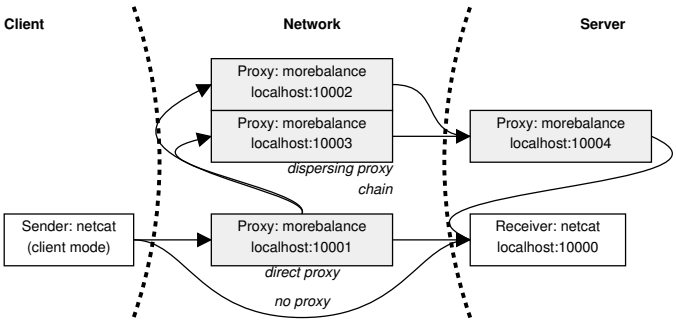


Fig. 6. Traffic forwarding choices with MoreBalance

In order to get a software which can serve as entry node, intermediary node and exit node of a dispersal communication resource, we have extended MoreBalance. It is a decade-old traffic forwarding and load balancing software written in C[1]. Each link can be a TCP stream or a UDP datagram destination, encrypted with TLS and compressed according to port modifiers in the configuration. Each listening link can be restricted to local interfaces. We have extended the software into a MIMO-Proxy (multiple inputs, multiple outputs) which are multiplexed internally for traffic flows in both directions. The setup is shown in Fig. 6. The corresponding configuration syntax reads: `10001:local use localhost as 10002 and localhost as 10003 with dispersion; 10002:local and 10003:local use localhost as 10004 with dispersion`. The dispersion implementation currently performs simple bit splitting. Alternatively, `chunking` and `replication` are already implemented.

In order to find out possible performance hits when using dispersed communication, we transmitted all-zero data blocks

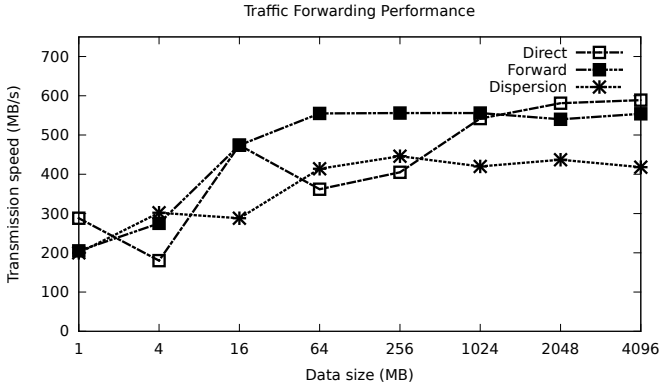


Fig. 7. MoreBalance traffic forwarding performance

of varying sizes with the standard utility `netcat` running in both client and server mode. In Fig. 7, the resulting performance graphs for proxy-less transmission, direct proxy and dispersed proxy are shown. It can be inferred that for smaller amounts of data, the proxy overhead can be ignored whereas for larger amounts there is a distinct, although not practically limiting, performance overhead of about 6.3% for just fowarding and 40.9% for the entire dispersion chain. The latter results can be optimised when considering the currently performed per-byte splitting into bitfields.

### B. Dispersed Storage Software

An overview about multi-target storage controllers is given in [9]. It concentrates on Cloud storage services although many implementations also work with other targets including local media.

In our previous work, we have designed and implemented NubiSave, a distributed storage controller[2]. NubiSave works in dynamic Cloud environments by evaluating storage service descriptions and defining optimal placement strategies for dispersed data fragments based on the non-functional property specifications within the descriptions. Its implementation is based on chains of filesystem modules through which the data flows with content-agnostic modifications such as compression or encryption. Now, we are working on an event stream processing module with filesystem interface which inspects the data and makes on-the-fly calculations and correlations possible, assuming the presence of dispersed computation software.

### C. Dispersed Computation Software

We are not currently aware of any software which specifically performs calculations and other operations over dispersed data. Therefore, we have created a prototypical implementation of both string search and integer addition algorithms. The implementations use Python and consists of a number of algorithm-specific modules and a rather generic client/server combination which uses the remote procedure call framework

[1]MoreBalance proxy: http://morebalance.coolprojects.org/

[2]NubiSave storage controller: http://nubisave.org/

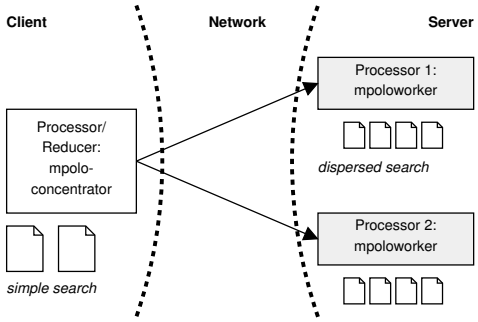


Fig. 8. Dispersed search with a specific text corpus and application

Pyro. As a search task, represented by Fig. 8, we have defined to look up the word *Polo* in Jules Verne's book *I Viaggi di Marco Polo*. Fig. 9 compares the function call performance of Python's native string search with the overall performance (including network transmission) of dispersed search which handles multiple data sets either serially or, through a configuration switch, in parallel with multithreading. The connection between client and server was changed from a local loopback device to a 1000 Mb/s Ethernet LAN, a 150 Mb/s WLAN network and a 2 Mb/s DSL WAN connection. The server tools were running on a virtual Linux machine with four Intel Xeon E5620 CPUs clocked at 2.40 GHz. It is interesting to observe that as the connection becomes the bottleneck, the inherent performance degradation of the dispersed search implementation becomes relatively minor due to not having to perform a full data transfer as with a conventional search. With larger datasets and optimised algorithms, this gap can become smaller and may even reverse.

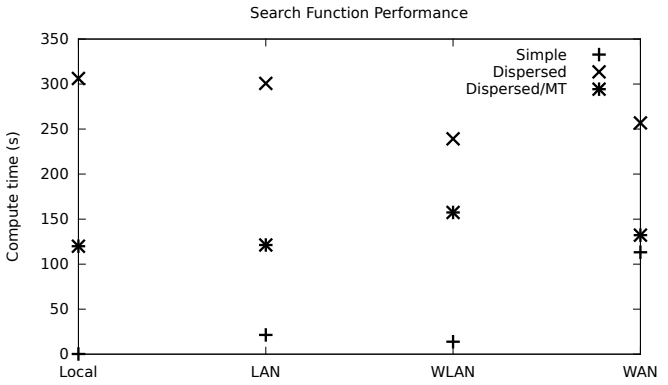


Fig. 9. Distributed search performance with and without dispersion

In Fig. 10, we would like to point out the statistical variety of the measurement results, taking the wireless connection as example. More divergence can be expected depending on the resources provided to the virtual machine, which is generally hard to predict in Cloud environments.

## IV. Research Outlook

Dispersed Cloud Computing is a promising new paradigm which enriches the varieties of parallel and distributed com-

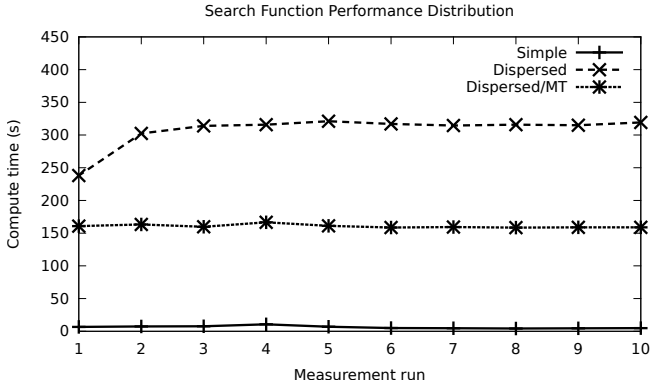


Fig. 10. Performance measurement stability: WLAN

puting. It is suitable for the major infrastructure services offered today, including compute, storage and communication services. Flexible generic software implementations are already available for both storage and communication, but so far only specialised software for dispersed compute services exists. The prototypes allow for future experiments with more complex scenarios which together with more useful compute abstractions at the programming level can be considered the most interesting path for further research.

## Acknowledgements

This work has been partially funded by the German Research Foundation (DFG) under project agreement SCHI 402/11-1.

## References

[1] S. Jha and D. S. Katz, *Applications for Distributed Applications and Systems: A Computational Science Perspective*, ser. Wiley Series on Parallel and Distributed Computing, 2014, no. 1.

[2] S. Kächele, C. Spann, F. J. Hauck, and J. Domaschka, "Beyond IaaS and PaaS: An Extended Cloud Taxonomy for Computation, Storage and Networking," in *2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*, December 2013, pp. 75–82, Dresden, Germany.

[3] D. Kelly, "A Taxonomy for and Analysis of Anonymous Communication Networks," Ph.D. dissertation, Air Force Institute of Technology, March 2009.

[4] M. Malensek, S. Pallickara, and S. Pallickara, "Polygon-Based Query Evaluation over Geospatial Data Using Distributed Hash Tables," in *2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*, December 2013, pp. 219–226, Dresden, Germany.

[5] S. Rao, R. Ramakrishnan, A. Silberstein, M. Ovsiannikov, and D. Reeves, "Sailfish: A Framework For Large Scale Data Processing," in *Proceedings of the Third ACM Symposium on Cloud Computing (SoCC)*, no. 4, October 2012, San Jose, California, USA.

[6] A. Brito, A. Martin, C. Fetzer, I. Rocha, and T. Nóbrega, "StreamMine3G OneClick - Deploy & Monitor ESP Applications With A Single Click," in *Fourth International Workshop on Parallel Software Tools and Tool Infrastructures (PSTI)*, October 2013, pp. 1014–1019, Lyon, France.

[7] M. O. Rabin, "Efficient Dispersal of Information for Security, Load Balancing, and Fault Tolerance," *Journal of the ACM*, vol. 36, no. 2, pp. 335–348, April 1989.

[8] Y.-D. Lyuu, *Information Dispersal and Parallel Computation*, ser. Cambridge International Series on Parallel Computation, 2004, no. 3.

[9] D. Slamanig and C. Hanser, "On cloud storage and the cloud of clouds approach," in *The 7th International Conference for Internet Technology and Secured Transactions (ICITST)*, December 2012, pp. 649–655, London, United Kingdom.