Federated Data Processing and Exchange at Global Scale

Michael Duller Gustavo Alonso
Systems Group, Department of Computer Science, ETH Zurich
8092 Zurich, Switzerland
{michael.duller, alonso}@inf.ethz.ch

1. INTRODUCTION

In the last decade the continuing growth of the Internet has enabled the interconnection of hundreds of millions of computer and communication devices all over the world. Simultaneously, a wide variety of applications and services has emerged to process and exchange data using technologies like web servers, HTTP, and web browsers.

However, the way technology and the usage of this data is evolving, is creating a conflict between user needs and existing data processing architectures. First, the web is no longer a vehicle for message exchanges and sharing static content. Web services, e-commerce, and Software as a Service (SaaS) ideas have resulted in large data processing engines being connected to the Internet as nodes of highly distributed applications. Second, the number of mobile and small devices with networking capabilities is rapidly increasing. Devices like smartphones, PDAs, smart toys, or sensor nodes have become advanced enough to take part in the Internet. Third, the amount of dynamically created and updated content is steadily growing. Dynamic information is both created by people and by devices. In many cases, the usefulness of this data is often inversely proportional to its age and its important to forward it to the user as soon as possible.

When taken all together, these developments raise very difficult challenges and even introduce substantial limitations from the user's point of view. Current services, cloud infrastructures, SaaS, and social networks are, for the most part, highly centralized, impossible to interoperate, closed, and to a certain extent static. In addition, there are obvious issues of privacy, access control, and customization that cannot be overlooked in the long term.

In this short paper we present *XTream*, a platform for developing open data stream processing overlays. With XTream, developers and even end users can easily program complex data processing meshes by composing distributed resources and processing steps. These meshes can be deployed at Internet scale (we have a first prototype running on PlanetLab), in clusters (we have a distributed data stream processing engine running standard benchmarks like Linear Road [1]), and be combined with arbitrary data sources (e-mail, text messages, RSS feeds, sensor data, etc.) and end devices (mobile phones, PDAs, or desktops). XTream offers a complete application model that treats data processing, data storage, and data communication as orthogonal concerns. This application model is supported by a lightweight middleware infrastructure that hides the complexities of communication, deployment, interoperability, and data management from the developer.

2. CHALLENGES

The main challenges for a platform like XTream are *dynamism*, *heterogeneity*, and *federated operation* at a global scale.

We address the problem of dynamism by treating all data as data streams. Data streams are very well suited for capturing data from mobile devices, devices that are only periodically connected, mobile sources, and also for defining the communication between distributed processing steps and end consumers. Data streams can also be seen as packet forwarding systems, which is of great help to cope with disconnected operation, high load variations, parallel processing of large data volumes, and flexible routing of the data flows.

We address heterogeneity by imposing a rigid but open and extensible programming model where all entities of the system follow well defined APIs, hiding their implementation details. XTream is capable of mixing processing elements written in different programming languages and of combining a wide variety of data management engines. The APIs are rich enough to support sophisticated interactions such as push or pull of the data, callbacks to data sources, and dynamic reconfiguration.

We address federated operation through a strict separation of data storage, data processing, and data dissemination. In XTream, there are well defined entities in charge of each one of these tasks. Programming a complex data processing and dissemination mesh is done by composition of these entities. There are no constraints on how these entities are deployed but the strict separation greatly facilitates implementing access control, system boundaries, ownership, performance optimization, integration of sources and end devices, deployment as a flexible overlay (including deployment on computing clouds), and supports the necessary scale-out.

XTream builds upon many ideas from peer-to-peer, overlays, data processing, data streaming infrastructures, data flow languages (e.g., Dryad, Hadoop, MapReduce), and Web 2.0 ideas. A very important component of XTream is the software engineering aspects intended to facilitate application development. The key contribution of XTream is that it is a platform for developing applications that combines all these ideas rather than focussing on the advantages of a single approach.

3. XTREAM: PLATFORM AND MODEL

To tackle the challenges of applications that federatively process and exchange data we propose *XTream*, a platform and a model for these applications. At the heart of XTream lies the model, which generalizes the data stream processing

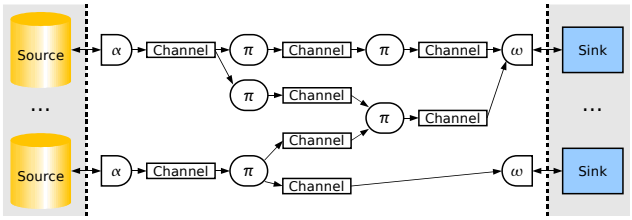


Figure 1: Data processing model of XTream

model beyond traditional streaming applications to encompass almost any application that exchanges data in a push or pull manner. The model is defined at different layers of abstraction ranging from a high level application building model down to an implementation model (see Figure "Layered details of abstraction" in the poster). In addition, we implemented a prototype of the XTream runtime platform, which implements the model and is continuously revised and extended.

3.1 Data Processing Model

Figure 1 illustrates the data processing model, which defines *channels* and *slets* (short for *streamlets*) as its entities. Three kinds of slets process data (π-slets, depicted as ovals) and adapt data sources (α-slets, depicted as half ovals) and data sinks (ω-slets, depicted as half ovals). Slets can be parametrized to allow for easy customization and reusability of the same kind of slet in different places. Channels buffer and forward data between slets and thus decouple producing from consuming slets. Slets can access data from a channel in a push or pull manner. In the resulting mesh of slets and channels, data is processed as it flows from sources on the left to sinks on the right.

3.2 Implementation Model

The implementation model (see Figure "Implementation model" in the poster) specifies how components interact with each other. XTream is designed as a SOA. Every individual component provides well defined services, which are used to interact with it. In the figure, services involved in data exchange with other components are depicted as thick, red bars. In addition, every component also exposes a management service that allows to parametrize, start, or stop the component.

In addition to channels and slets defined by the data processing model, the implementation model introduces *connectors* as third entity. They form a level of indirection between channels and slets and provide for access to channels in a remote framework. Thus, they incorporate the communication aspect in the model.

Access to a remote channel is realized through a pair of matched connector halves, of which one is registered on the framework where the channel resides and the other in the framework where the channel is accessed from. The latter part buffers data as proxy for the channel to provide local buffer access for connected slets and allow for optimizations like proactive or reactive caching. Every pair of instances of XTream frameworks that interact with each other maintain a single connection through which they exchange management information as well as data that is flowing through remote connectors.

3.3 Prototype Implementation

The prototype of the XTream platform is implemented using OSGi [5] and Java. Channels, slets, and connectors and the implementation of the XTream runtime platform itself are designed in a modular way and make heavy use of loose coupling through dynamically bound services. This provides for flexibility and extensibility of the platform and facilitates development and management of the system.

Despite the fully dynamic binding of services, the implementation incurs hardly any overhead on the data path, as the service objects are proactively cached using a service tracker. Additionally, the expression over service properties used in the service tracker is always of constant length and can be constructed solely from a component's local properties, which is illustrated in Figure "Component interaction" in the poster and helps both consistency and performance.

4. PROJECT STATUS

One main application area we are looking at is personal information processing. After initial experiments in this area, which we have successfully demonstrated in [4], we have defined a first model and implemented a prototype of the platform. Recently, we reported on the status of personal information processing in [3].

To test XTream's ability to cope with a widely distributed setup of potentially unreliable nodes, we deployed a synthetic application that simulates personal information processing on 200 PlanetLab nodes. PlanetLab is an ideal testbed for XTream as it provides us not only with Internet scale distribution but also with an environment that is very similar to people's personal computers in terms of other processes contending for resources and spontaneous shutdown or crash of individual nodes. We ran the application for 24 hours and concluded from the results that XTream can cope well with such a dynamic and widely distributed environment.

At the moment we are evaluating the system under the Linear Road benchmark, which represents a conventional stream processing workload, to analyze the overhead incurred by the platform. We chose the implementation of [2], wrapped their MXQuery engine as an slet and their storage implementations as channels, and rebuilt the same workflow they used. Our implementation was able to sustain the same load factor like the original implementation, as XTream added only little overhead.

5. REFERENCES

[1] A. Arasu, M. Cherniack, E. F. Galvez, D. Maier, A. Maskey, E. Ryvkina, M. Stonebraker, and R. Tibbetts. Linear Road: A Stream Data Management Benchmark. In *VLDB*, 2004.

[2] I. Botan, P. M. Fischer, D. Florescu, D. Kossmann, T. Kraska, and R. Tamosevicius. Extending XQuery with Window Functions. In *VLDB*, 2007.

[3] M. Duller and G. Alonso. XTream: An Open, Distributed Platform for Processing Personal Information Streams. In *WDDDM*, 2009.

[4] M. Duller, R. Tamosevicius, G. Alonso, and D. Kossmann. XTream: Personal Data Streams. In *SIGMOD*, 2007.

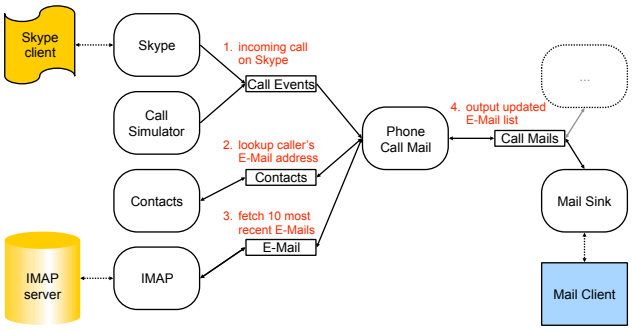
[5] OSGi Alliance. OSGi Service Platform. `http://www.osgi.org/`.

Federated Data Processing and Exchange at Global Scale

Michael Duller, Gustavo Alonso / Systems Group / Department of Computer Science / ETH Zurich

Application: Integration of VoIP and E-Mail

- display last E-Mails exchanged with person talking on Skype
- plain and distinct application that covers a set of key challenges



Application: Friends' Recent Photo Exchange

- different access patterns (push/pull)
- aggregation of all photo feeds into one

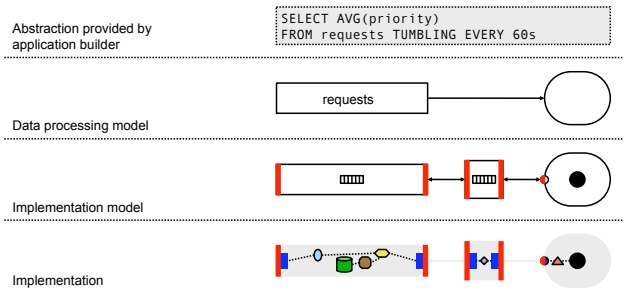
Goal

Programming model and runtime environment for

- processing and integration of dynamic and heterogeneous data
- easy implementation of simple as well as sophisticated applications
- federated operation of distributed, independent applications

The XTream Approach

- generalization of data stream processing model
- separation of processing (slet), storage (channel), and communication (connector) concerns into distinct elements of the model
- direct interaction between peers



Today's Options for Data Processing and Exchange

Database / data stream engines
- big and complex
- limited set of operators

Peer to peer systems
- focus on data exchange
- limited/no processing model

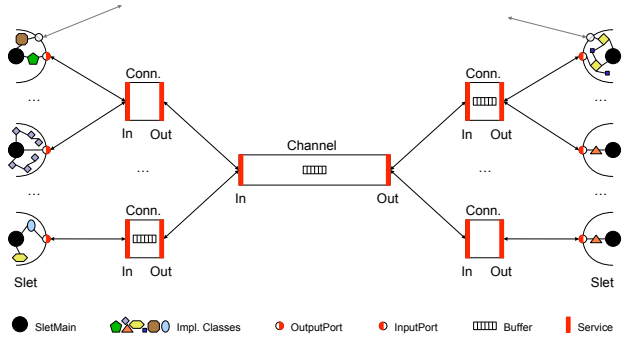
Cloud Computing, SaaS
- bring data into the cloud
- transfer results back

Data Parallel Cluster Computing
- batch job scheduling
- restricted processing model

➔ unapt for small, local applications processing dynamic or push data
➔ unapt for connecting independent instances of these applications

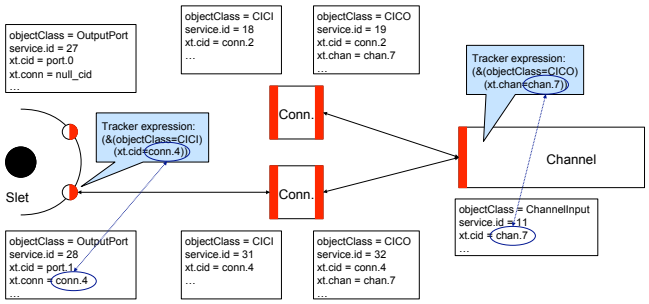
Implementation Model

- component interaction through services
  - dynamic binding at runtime
  - well-defined boundaries
- connectors as additional level of indirection
  - proxies for remote channels
  - windows (buffers) for connected slets



Prototype: High-performance, Dynamic Component Interaction

- proactive tracking of service objects of interest
- no overhead (e.g., lookup) during normal operation (data path)
- incremental overhead if bindings actually change
- tracker expression
  - always of constant length
  - created solely from component's local service properties



ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

inf | Informatik
Computer Science