mlstate

Opa

The Cloud Language

Introduction

Opa key features:

- ✓ Rich user experience
- ✓ Sane development model
- ✓ Robust foundations
- ✓ Streamlined tools
- ✓ Extreme security
- ✓ High performance
- ✓ Built-in scalability
- ✓ Simple to program

For all users, whether mobile or sedentary, the Web is now the platform that matters. Users need applications they can access from all their devices, applications that can share information in real-time, regardless of the operating system, and deliver a rich user experience. Less visibly, users also need their applications to protect their data, its integrity, their privacy. The needs of editors are different: they need applications that scale up automatically to handle countless users, that can be administered easily, and that survive both hardware failures and attacks, no matter how complex. Finally, developers need powerful, simple, streamlined tools, that allow them to concentrate on the features, rather than having to reinvent the wheel countless times - and fix the security of said wheel in emergency on a regular basis.

The Opa platform has been designed to deliver all of this, and more.

Opa is composed of an exclusive core technology, a programming language designed specifically for the web, a library of functions and components tailored to servers and browsers and a set of programming tools built to improve both the productivity and experience of developers and the quality of resulting applications.

Opa is based on the results of dozens of years of fundamental research and 80+ man·year of R&D. It condenses numerous scientific and technological breakthroughs in terms of programming languages, security, distribution, data management, scalability, in one simple package, delivering the One-Pot-Application. These advances not only make the infrastructure of applications more reliable, absolutely removing the possibility of a number of attacks and diminishing the impact of others, they also make development simpler and safer thanks to a new paradigm, by automatically detecting numerous potential security holes during the development phase, and by automatically taking care of most of the tedious and unproductive aspects of application development. On top of this, Opa provides a comprehensive library of functions and components that can be easily and quickly composed into complete web applications.

Opa combines simplicity, safety, security and scalability. For the web, for the cloud, today.

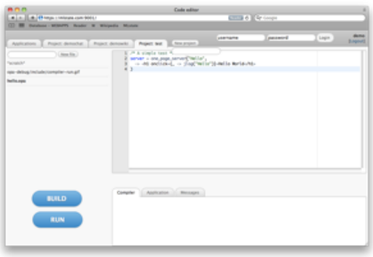
Opa for enterprise communication tools

1. Rich applications

Opa is designed specifically for the web, and on the web, Opa is king. Opa has been used to develop web-based productivity tools, B2B and B2C e-commerce applications, but also single- and multiplayer games, enterprise communication tools, social networks, development platforms, Facebook applications...

Users can access these applications with Windows, MacOS X, Linux, but also from iPhones and iPads, from Android phones and tablets, or generally, from any modern device with a modern browser. They need no installation, no plug-ins, no extension. All it takes is one hyperlink, or one tweet, or one Google search and the application is on.

User interfaces: Great looking and extensible



```
/* A simple test */
server = one_page_server("Hello",
  -> <h1 onclick={_ -> jlog("Hello")}>Hello World</h1>
)
```

PaaS tools for developers, in Opa, for Opa.

Opa provides a rich library of user interface components, from tables to lists to smart date display, from weather controls to tabs to city maps. Straight out of the box, these components work – and they look great. Of course, your application can also be customized to match the visual identity of your company. And should you need to, adding new components is completely straightforward – just group a few existing components or write your own XHTML and CSS for the display, and add some Opa code to customize the interactivity or the database access. No need to write JavaScript code.

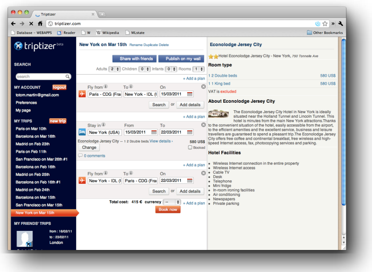
Of course, the user experience of modern web applications requires more than just components. Nowadays, the web requires high interactivity, animations, special effects. And since a number of devices geo-localization, access to the touchscreen, etc., the rich library of Opa can take advantage of such features.

Real-time web

Opa for multiplayer games

Today's web is real-time. Users want to be informed immediately when their friends connect, or when a new mail has arrived, or when their adversary has just played. System administrators need to know at once that a hardware problem has occurred, or when suspicious activities arise. Site owners need to take decisions immediately, to rent or release resources in real-time.

Real-time web lies at the very core of Opa. This means that, out-of-the-box, Opa applications can communicate, between the browser and the server, between two users, or between two servers. Communications can be explicit in the application code, or can take place behind-the-scenes, and can serve to send or receive any kind of safe data, including text or events. With a few lines of code, it is even possible to write applications that can be remote-controlled from a server, or fully reconfigured without any reload!



Opa for Facebook-enabled B2C

Social web: Connected

Today's web is social. So is Opa.

Opa applications can contain forums, blogs, newsfeeds, chats. Or, they can connect to Facebook or Twitter. Opa-built applications have no limits and can interact with other networks, applications, databases, thanks to Opa support of the open standards SOAP, REST and WSDL.

Your device: Handled

Internet Explorer	≥ 8
Firefox	≥ 3.5
Chrome	≥ 6
Safari	≥ 4
Opera	≥ 10

Compatibility with major web browsers

Are your users browsing with Safari? With Internet Explorer 9? Perhaps with Firefox Mobile? Then they can use Opa applications. Actually, any recent browser will be able to execute Opa applications flawlessly[1].

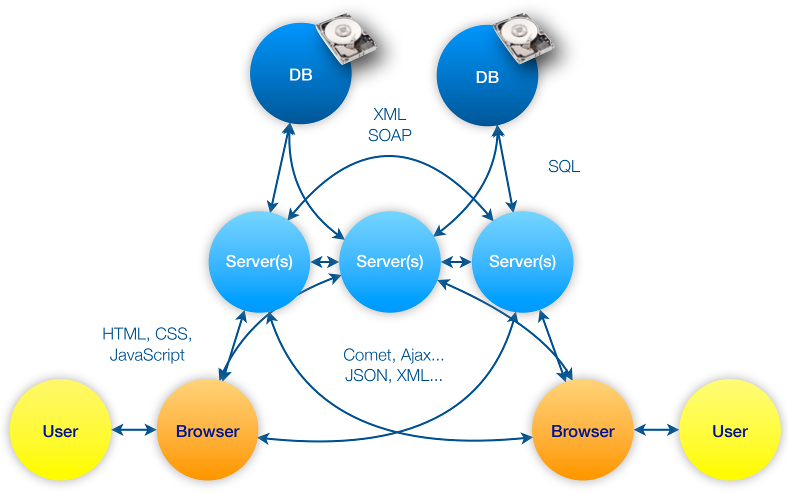
Of course, the user won't need any plug-in, or any installation. In particular, Opa does not need the Flash plugin and is therefore compatible with mobile devices like the iPhone or iPad. That's because Opa is based on established and open standards.

2. Web development made sound

In today's Web 2.0, competitive online applications are feature-rich. Web 2.0 applications need high interactivity, real-time notifications, a deep visual interface. Invisible features must also be rich, to ensure security, allow transparent maintenance or recovery from hardware failures. Of course, all these features can be implemented on most platforms, given a team of experts, clear specifications, the right tools, and enough time. Unfortunately, most Web 2.0 developments fall short of their goals, victims of complexity overload, and deliver incomplete applications, lacking in security, or with features that just can't work.

This is where Opa can help you: a technology designed from the ground up specifically for Web 2.0 applications, to set you free from complexity overload.

[1] With the exception of Opera Mini. This browser is designed specifically for devices that are not powerful enough to handle interactive applications.



Architecture of a typical simple web application without Opa.
A dozen of distinct technologies and 7 distinct paradigms.

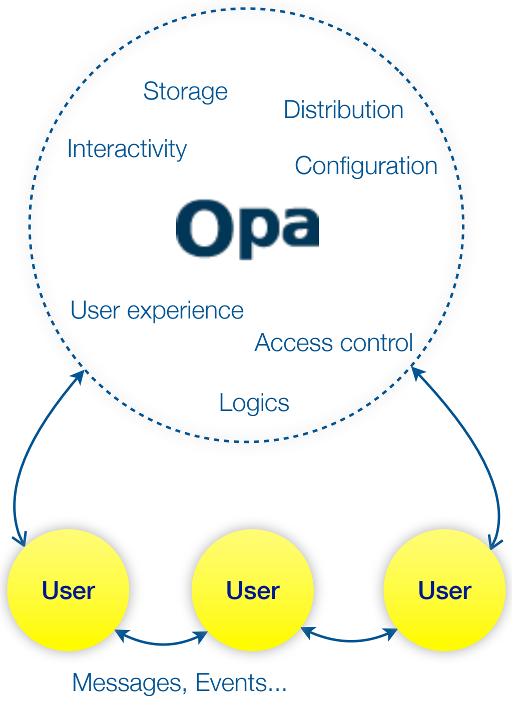
JavaScript? SQL? Comet? Ajax? JSON? You won't need them

Web applications are always distributed, no matter which technology has been used to build them: part of the application is executed on a server, in particular the construction of the pages, part of the application is executed in the database, and part of the application is executed in a web browser (or "web client"), in particular the display. That's three sides, programmed with three different paradigms or more, and communicating with at least three distinct technologies. With most web platforms, the three sides need to be developed essentially separately, three different languages or more, and only then connected into a hopefully consistent whole. If the design or implementation are incorrect, the application will fail – sometimes spectacularly, or more often in subtle and hard-to-debug manners.

In Opa, distribution between client and server is fully transparent. The developer only needs to care about users, not about browser instances, low-level requests, serialization formats, COMET or AJAX. From one source code, the compiler determines which code requires the features of the server, which code requires the features of the client, and decides automatically where feature-independent code will be executed. The paradigm of sessions behind Opa lets you easily share data between users and between users and server, or even between servers, while ensuring that code executed on a client will not interfere with the code executed on another client – that is, of course, unless you wish them to interact.

This level of transparence extends to the database. Developers only need to care about data and how they intend to use it, not about serialization, concatenation of requests, or mapping of user access policies to database access policies. From one source code, the compiler determine which code requires the features of the database, and constructs the corresponding application. In Opa, manipulation of data is just that simple. And if specific data access policies are necessary, they can also be implemented in Opa, using simple common mechanisms to ensure the security of data, resources or entry points.

Wherever security or performance needs of a specific application dictate that some code should be executed specifically on one side rather than on the other, developers only need to add annotations, which the compiler will automatically follow when placing code. The compiler automatically ensures that no entry point is left unguarded – a vector that malevolent users could exploit to attack the application.



The same application in Opa. Only one language and one technology.

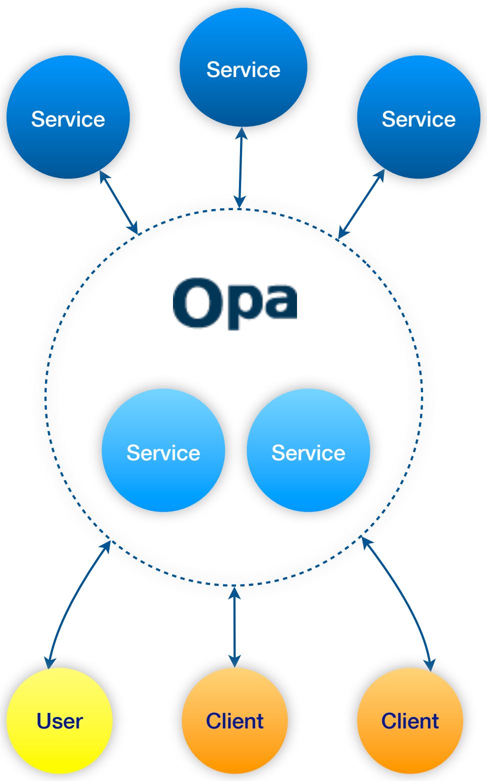
Service-Oriented Architectures? Native

One of the most interesting aspects of web applications is the ability to integrate heterogeneous and distant web components through service-oriented architectures, using protocols such as SOAP and REST.

Opa applications integrate seamlessly either as client or server for SOAP and REST services. Opa transparently parses XML requests and produces XML based on replies. Services implemented in Opa respect industry standards, can use all the features of Opa, including Opa's distribution scheme and Opa authentication, and can be either stateless or stateful, asynchronous or synchronous. Wherever needed, this mechanism can easily be extended to other service-oriented architectures.

Browser compatibility? Automated

By definition, web applications are cross-platform. However, any web developer can confirm that this compatibility has a cost. Most of the time, an application written with Safari in mind will fail with Internet Explorer at first, and once it works with both Safari and Internet Explorer, it still needs to be debugged and partly rewritten for Chrome, and again for



Firefox.

Opa in a Service-Oriented Architecture

Not so with Opa. Opa applications just work with all modern browsers. In addition, the code generated by Opa is generally much smaller and much faster than anything a developer would write directly when programming in JavaScript.

Deployment? Seamless

For most web applications, deployment is a period of nightmares, a time spent configuring nodes, database servers, remote storage, setting up and testing layer after layer of security policies, in the operating system, the web server, the application, the database, the middleware, the administration tools, and layers again and again in the virtualization environment... Most distributed storage mechanisms even require the setup of dedicated disks and file systems, if not dedicated proprietary hardware.

Opa applications are designed to be deployed simply. Just launch the application on each node you wish to use, and let them communicate. That’s because Opa applications are tierless: one technology, one paradigm, no middle layers that get in the way of development, deployment and performance.

3. Web development made secure

The first layer of protection against attacks and accidents is your infrastructure. Behind the scenes, the Opa platform provides an impressive level of support for safety, security and privacy.

Web server: lightweight and secure

At its core, every Opa application is based on the One-Pot-Application Server, or Opak, an exclusive web micro-kernel that handles resource management, low-level and high-level network communications, concurrency and interactions with the Operating System. By opposition to most web servers, Opak has been designed from the ground up with provably safe-by-design techniques and implemented using a tower of memory-safe Domain-Specific Programming Languages.

This discipline has led to a kernel that not only is extremely safe and very flexible, but also provides one of the smallest Trusted Computing Bases in the industry. In effect, Opak is one of the only web server cores fit for EAL4+ certification.

Loose bindings: secured

Implementing web services and applications is a task which commonly requires the usage of numerous different technologies, for client-side displays, server-side computation, database, security, communications, distribution, etc.

These technologies are generally loosely bound by hand-written protocols often based on approximative specifications,

✓ SQL injections: immunity

✓ XSS injections: immunity

✓ Attacks on semi-structured data: counter-measures

using whichever tools are at hand. This means that, even when each technology is sufficiently safe and secure, the protocols themselves can be targeted by attacks. Indeed, Google itself has been victim of “XSS injections” – an attack targeting protocols between the web client and the web server – in 2007, 2008, 2009, 2010 and 2011, just as Twitter, Amazon and most other companies present online have often proved vulnerable to such attacks. Consulting firm WhiteHat Security estimates that, in Q1 2009, 65% of the web sites were vulnerable to such attacks, and 18% to “SQL injections” – a comparable intrusion technique targeting protocols between the web server and the database. Opa applications are totally immune to such attacks and to numerous other attacks on protocols: its unique database compilation technique ensures that no forged request can ever reach the database, while its compiler-enforced type-safe systematic validation of data at system boundaries guarantees both that no client-side code can be injected in the application and that the structure of messages between client and server cannot be deformed by malicious third-parties.

Library: secured

✓ Buffer overflows: immunity

✓ Format string attacks: immunity

✓ Attacks on low-level libraries: counter-measures

All applications rely on a standard library of functions and data structures, and that’s true not only of the web applications you intend to build but also of the infrastructure upon which they are themselves built. While careful developers will ensure systematically that they use library functions safely, lack of documentation, misunderstandings or human errors can open the door to countless attacks, including buffer overflows or underflows, format string or null-byte injections – occurrences so common that microprocessor and operating system designers integrate increasingly complex counter-measures to their respective products in order to mitigate the effect of successful attacks. Opa applications and the Opa runtime environment itself, being written in a memory-safe programming language, are immune to these attacks. Furthermore, use of the Opa standard library by developers is itself protected by strong type-safe guarantees, which safeguard applications against most misuses of data structures.

Further counter-measures

✓ Denial-of-Service: counter-measures

✓ Man-in-the-middle: counter-measures

Malicious users may try to bring down your application by saturating it with requests from thousands or millions of pirated computers. For this reason, Opak integrates counter-measures against such Denial-of-Service attacks, discarding requests it can’t serve, identifying likely attackers – or letting you identify them, if you prefer – and throttling their requests. Opak also provides transparent load-balancing between web servers, between database servers and between computation servers, designed to ensure scalability to high loads, and works well with established solutions for detecting and defending against Distributed Denial-of-Service attacks.

Of course, should your your organization decide to implement a defense-in-depth strategy, this is also possible with Opa. The platform offers all the necessary features to complement its strong perimeter-defense with any technology you wish to use behind the scene, whether they're based on Opa or not.

Database: safe

✓ Archives: safe

✓ Latest version: safe

Sadly, despite your best precaution, cunning attackers, misinformed users or distracted employees might manage to compromise your data and introduce false information. Or perhaps the computer holding the data experiences a hard-drive failure which makes it unusable. The Opa database was designed to cope with such situations. Its unique incremental update system makes sure that you can always roll back your data to a previous version, either manually or from your application, without wasting the disk space which would be necessary for full back-ups. In addition, the Opa database will automatically replicate your data to several computers, and use all the copies to speed up transactions or survive hardware failures without a glitch.

This is not the only layer of protection provided by the database. Thanks to its unique mechanisms of data sets and virtual access, any Opa application can plug invariants into the Opa high-level database and monitor database access, ensure that values remain consistent with respect to each other, that no data entry can render the state invalid, etc. This mechanism is probably the most powerful validation mechanism ever seen in an industrial database.

Paradigm: safe

✓ Informations: protected

✓ Security invariants: protected

✓ Concurrency: safe

✓ Distribution: safe

While web applications and services are usually designed to be mostly stateless, the tools and languages used to implement these applications are typically designed to program with states. A direct consequence of programming with such paradigms is that it is quite easy for a programmer to accidentally break a safety or a security invariant introduced by another programmer, or to write code that will fail for mysterious reasons when executed in a parallel environment, just as it is easy to leak information between unrelated modules, and from these modules to the user. Indeed, WhiteHat Security estimates that 47% of web sites leak informations involuntarily – exactly how many such leaks are related to the use of an inappropriate paradigm is unknown. By opposition, the paradigm of sessions in Opa makes it easy for developers to write programs that do not depend on state, and, whenever states are required, to protect them with well-specified protocols, hence making sure that other developers can't break invariants by accident. This same paradigm limits the possibility of leaking informations accidentally, and guarantees that your application will execute even better on several cores, several processors or several servers.

Static checks: safe

✓ Resources: protected

✓ Concurrency invariants: protected

✓ Database: protected

The Opa compiler automatically performs checks on manipulation of data, through a unique static structural type system with inference. This means that, while developers can write their code either as if they were using a dynamic language – or as if they were using a traditional static language, if they prefer – numerous properties are automatically checked, many more than in most languages, including Python or Java. In particular, developers can add type annotations and type abstractions to ensure that breaches to high-level properties are detected during compilation. This is used in Opa's standard library, for instance, to guarantee that, if your application requires authentication, then each of your resources, pages, images, etc. is either explicitly marked as public or protected by an authentication check. Furthermore, this guarantees that all authentication checks in an application agree to use the same set of authentication mechanisms. The same kind of check ensures that developers cannot accidentally introduce vulnerabilities to Cross-Site Scripting injections.

Strong structural type-checking is also used, still transparently, when communicating between clients and servers, between users, between servers, or with the database, to ensure that all parties share the same definitions of structures. This is used in the database, for instance, to ensure that if a field is expected to never be empty, then any attempt by the program to make that field empty will be detected during compilation.

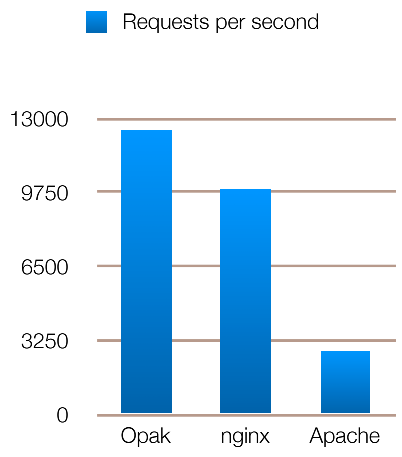
Further checks

✓ Ajax: checked

The Opa compiler doesn't stop its checks at data manipulation. It also detects a number of cases you may have forgotten to handle in your application, or cases you handle but that can't happen. Just as importantly, it will examine how your programs are distributed between web browsers and servers, to determine automatically if your distribution scheme can open security holes that could be exploited by malevolent users to access or alter confidential informations.

4. Web development made scalable

Performance: impressive



Benchmarking Opak, nginx and Apache.

The Opak micro-kernel is not only small, secure and flexible, it's also designed for performance. And this design has proved good, since Opak is one of the fastest HTTP servers in the world.

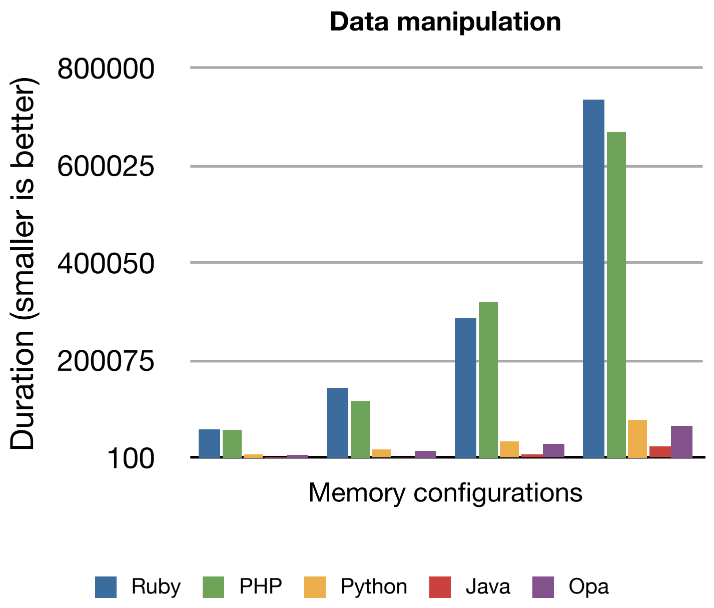
This doesn't stop with the HTTP server. Opa applications are fast, too, and they keep getting faster. Benchmarks indicate that Opa applications consistently outperform optimized Python, Ruby or PHP, and that Opa scales better than any of these languages.

Even further: Opa contains a state-of-the-art optimizer for client code, which ensures that this code is both fast to transfer, small enough to fit on memory-constrained devices, and fast to execute, even on relatively low-performance smartphones.

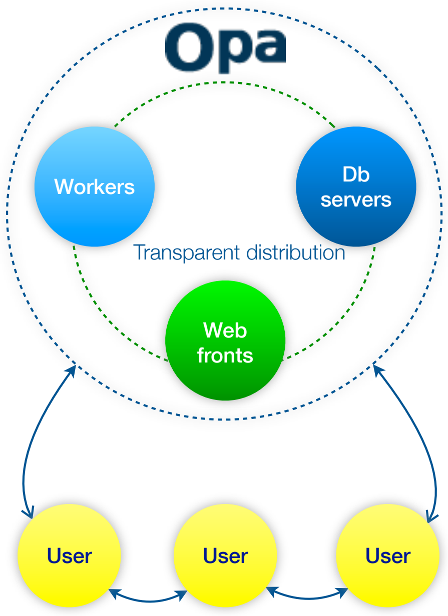
Data scaling: built-in

Most industry-grade web applications need to store large amounts of data, usually more than can fit on a single server. Furthermore, this data can be accessed on behalf of many users simultaneously, and needs to be replicated to ensure that hardware failures will not cause service outages.

In Opa, distribution and replication of data and requests is fully transparent, and can be further completed by an extremely simple high-level API. The administrator only needs to launch several instances of the application on several computers, and they will start communicating and working together. Any Opa executable can work as a web front-end, as a computation back-end (or "worker") and/or as a database server, depending on its configuration. Even better: new executables can be added while the application is running, without having to relaunch or reconfigure the servers.

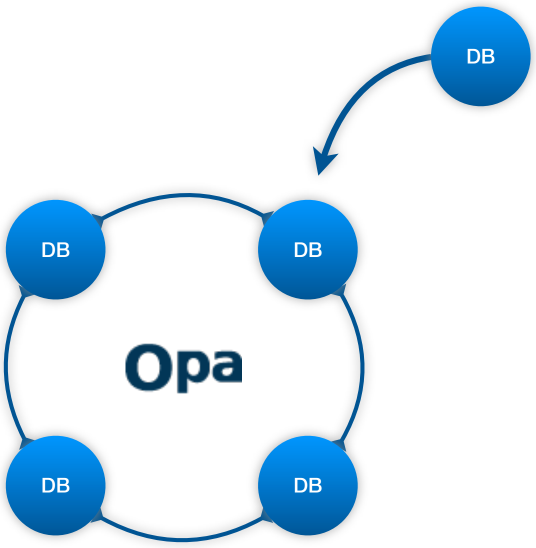


Depending on resources available, the Opa runtime will automatically dispatch connecting users to the most appropriate web front-end. The web front-end then executes user requests and launches database requests on behalf of the user. Database requests are executed transparently on one database server or more, depending on which server holds the required data, without blocking the web front-end. The Opa compiler also transparently optimizes requests so as to minimize the exchange of information between the web front-end and the database server.



Distribution in Opa is simple and safe

Developers may also decide that heavy computational tasks should be entrusted to specialized computational back-ends, by annotating sessions as distant. This operation integrates seamlessly in the Opa paradigm. Further optimizations are executed transparently to minimize the amount of communication required between computational back-ends and web front-ends. The resulting code is generally much faster than anything a developer could write manually in any



reasonable amount of time.

Need scalability? Nodes can be added, removed or replaced during the execution of your application.

To resist hardware failure, data is automatically and transparently replicated between several database servers. If a database server fails, the system can continue to operate, as long as needed, even while the server is being replaced. Administrators may configure the required amount of redundancy to tailor it to available resources and requirements, and if necessary perform fine-grained configuration on the redundancy of individual pieces of data.

Conclusion

Opa is a breakthrough in the realm of web applications, both for developers, editors and users. Not only does it make web development simpler and faster, but it also:

- handles most of the work on scalability transparently;
- handles most security issues automatically;
- empowers developers to deliver quickly feature-rich applications.

A. Appendix: Main standard library widgets and components

Core

Text input	Verified input	Image	Checkbox
Selection	Icon	Label	Button

Containers

List	Tree	Panel	Multi-panel
Tabs	Dialog box	Menu	Status bar
Calendar	Switch button		

Advanced input

Autocomplete	Login box	Search box	Password box
Wysiwyg editor	Multi-user editor		

Advanced display

Date printer	Text trimmer	Chart	Weather

Specific

Download	Upload	Date picker	Badge
Tag cloud	Tag list	Bread crumb	Color picker
Vote	Progress bar	Slider	Auto-scroller
Google Maps	Facebook connect	Twitter	

Special effects

Scrolling	Fade in/fade out	Fade to	Slide in/slide out
Animate to	Time-based effects		

About MLstate

MLstate was founded in 2007 by a former researcher to change the way we develop and run web applications using fundamental research on the safety of programming languages and the goal to bridge the gap between industry and academic research.

The company R&D team is now composed of over 10 PhDs and over 10 Research Engineers. MLstate is a member of the international competitiveness cluster System@tic since 2008 and a member of the W3C since 2010 and has partnerships with several prestigious research labs worldwide.

MLstate was finalist of Paris Innovation Grand Prix in 2007, cited in the 'Best of' Atelier BNP Paribas in 2008 and won the French Ministry of Research Award 2008. In 2010, MLstate was selected to join Oseo Excellence, the Top 2000 French fastest-growing companies.

Learn more at http://mlstate.com.