power and tools that allow the developer to create and compose abstractions while making use of the full power of concurrency and distribution.

1. Abstractions for data naming, sharing and storage (consistency and versioning), computation (processes), communication (invocation).
2. These abstractions can be instantiated and controlled by a separate control plane, enabling elastic configuration of the number and placement of computation and data entities, transparently to the program text.
3. A composition mechanism based on scoping enables information hiding.
4. Communication supports asynchronous (concurrent) and synchronous (consensus-based) operation invocation, with transactional and causal consistency guarantees.
5. Communication channels are based on a publish-subscribe/data flow model, with forward and backward paths, and carry any mixture of state, delta, or operation.

Furthermore, there is a fundamental tension between the requirements of correctness (no execution bugs, no loss of data) and performance (with metrics that include availability, responsiveness and throughput). Correctness requires events to happen in a reliable, deterministic way. Performance requires concurrent, asynchronous execution. The “right” solution to this trade-off depends on application requirements, environment, workload, resources, etc. Getting this right is difficult. Thus, non-expert programmers are required to navigate uncomfortable and weighty trade-offs in the presence of non-composable, non-deterministic, and weak consistency.

**Proposed research**

This situation needs to be improved with a high-level approach. We seek a doctoral student to develop methods and tools to support the programmer of distributed services, and to apply them to the incremental development of a representative service such as a distributed storage service, or a distributed indexing service.

Highly successful and explicative abstractions already exist, such as (at opposite ends of the spectrum) consensus or data flow. Frameworks and languages are making distributed programming easily accessible in some restricted domains, for instance MapReduce, Spark or TensorFlow. Our team has been active in research to provide both high availability and correctness, e.g., developing high-level distributed data types (CRDTs), and efficient and correct protocols (NMSI, TCC). Our CISE tool verifies that the execution model of an application is sufficiently synchronised to guarantee the application’s safety invariants.

However, currently, the heavy lifting remains manual. We propose to develop programming methods and tools that allow the developer to create and compose abstractions while making use of the full power of concurrency and distribution.

**Contexte et atouts du poste**

**Background**

Distributed computing is relevant to developers in many different application areas, for instance, mobile computing, Internet of Things (IoT), high-performance computing, Network Function Virtualisation (NFV), neural networks, or internet gaming. However, distributed programming remains difficult and error-prone, exposing users, the economy, and critical infrastructure to bugs and security violations. Indeed, concurrency and failures, essential features of a distributed system, are difficult to abstract away, and interacting concurrent processes do not compose well. Furthermore, any large-scale distributed system suffers undetectable failures and cannot be strongly consistent, i.e., processes cannot reach consensus on a single up-to-date view of shared state; this is a fundamental result of distributed system theory (the FLP and CAP theorems).

Thus, non-expert programmers are required to navigate uncomfortable and weighty trade-offs in the presence of non-composable, non-deterministic, and weak consistency.
6. Transparent storage and communication of metadata, such as timestamps, provenance, security labels, or accounting information.
7. Specifying preconditions, effects and invariants, in order to enable verification.

At the same time, our approach helps avoid many of the opportunities for error, by focusing on the essential properties of application correctness. It is often the invariants required over application data that dictates the protocol for accessing the data; this is an intuition that programmers commonly apply. Hence, we aim to apply leverage language and verification tools, to aid the programmer in choosing the best consistency level and in synthesising a program that respects its specification.

**Principales activités**
The research has both a fundamental and an applied aspect and aims for practical results.

**Compétences**

**Requirements and application**

Candidates to this position should hold a Master's in Computer Science/Informatics or a related field. They must be excited by research in systems, distributed systems, databases, and/or language and verification, and should have an excellent academic record in one of these areas. He or she will be developing and experimenting software at large scale. Teamwork and communication skills, industrial experience, and good knowledge of Erlang and/or node.js is a plus. This is offered as part of Inria's annual PhD competition. To apply, please provide the following information:

- A resume or Curriculum Vitae.
- A list of courses and grades of the last two years of study (an informal transcript is OK).
- Names and contact details of at least two references (people who can recommend you), whom we will contact directly.
- If relevant, a link to your publications and/or open-source developments.

**Avantages sociaux**
- Subsidised catering service
- Partially-reimbursed public transport

**Rémunération**

Gross Salary per month: 1 982 € the first 2 years and 2 085 € the last year