(a) Applications

What are the applications that motivate future systems?

- Mobile applications backed by cloud servers and storage have great potential for impact:
  - Big data computations on clouds.
  - Social media applications, e.g., media submission, archiving, and manipulation.
  - Healthcare diagnostics and monitoring.
  - Virtual and augmented reality.
  - Voice recognition.
  - Intelligent vehicles and highways.
  - Smart cities.
  - Internet of Things.
- Compared to more research-oriented applications, mobile applications can have a direct and immediate positive impact in people’s lives.

What advances in algorithms and programming systems will support these applications?

- Programming languages need semantics that allow transparent execution in networks, and enable formal reasoning about correctness of network-based implementations.
- New advances algorithms are needed for verification and testing of systems developed in distributed programming languages:
  - Advances in methods for reasoning about probabilistic systems such as Statistical model checking and Euclidean model checking.
  - Runtime verification and monitoring using a mix of deterministic and randomized algorithms to handle and react to both detailed low-level information and uncertain aggregated information about large-scale behavior.
- Mainstream programming languages need to be equipped with high-level concurrency abstractions close to the mental model of developers, not low-level primitives such as locks.
- Probabilistic concurrent programming languages that include continuous variables to simplify modeling the real world.
- High-level programming languages and associated platforms are needed for easy development and deployment of cloud-based mobile applications that leverage existing cloud-based services. Languages should not require extensive knowledge of concurrency semantics and low-level details in general.
(b) Systems

What hardware architectures and distributed systems will support future applications?

- To free application developers of concerns about low-level hardware and other infrastructure concerns, as well as the burden of resource procurement, application platforms (Platform-as-a-Service) must be provided and have evidence-based guarantees both in terms of correct execution and of non-functional requirements such as performance.
- Programmers cannot be expected to enunciate all application requirements at development time. Instead, distributed systems in which applications execute must be equipped with extensive monitoring and inference capabilities, so that they can adapt execution to changing requirements.

What challenges do we face in design and management?

- Formal methods techniques are important in the development Platform-as-a-Service and Software-as-a-Service solutions, yet are not widely applied in practice.
- Cloud platforms must be evolution-aware, i.e., monitoring and runtime verification must facilitate adaptation needed to support changes in applications.

(c) Technologies

What emerging technologies will change fundamental assumptions in hardware and software? What constraints disappear?

- Sensor networks and sensor clouds.
- Software-defined networks.
- New hardware will have approximate components that may silently introduce inaccuracies in computation in order to provide significantly improved performance and greater energy efficiency.
- Location transparency of computations and data (active and passive objects) is becoming more prevalent, lessening the need of developers to keep track of specific hardware platforms. Mobility of code, program state, and data is becoming the supported default rather than niche extension.

What challenges arise?

- Techniques and tools are needed to:
  - Assess tradeoffs between accuracy and energy consumption.
  - Determine sensitivity of computations (distributed and local) to perturbations caused by hardware with approximate components.
  - Dynamically adapt executing software to changes in the environment, e.g., hardware.
- Distributed systems must be able to utilize “asymmetric” computational hardware such as GPUs transparently, when available.
- Services traditionally provided by hardware are gradually taken over by software, e.g., networking and message routing. This increases requirements on software quality.
- Deployment and runtime concerns generally cannot be enunciated at development time due to evolving environments and underlying hardware. Consequently, software platforms must have considerable built-in adaptability.
(d) Methodologies

How should we perform interdisciplinary research that spans applications, systems, and technologies?

- Co-develop system implementations and programming languages with features such as formal semantics and formal correctness guarantees.
- Software frameworks can be synthesized from existing applications and environments rather than constructed in isolation, from scratch.
- The building process for software frameworks, hardware platforms and technologies can be facilitated use of synthesis and probabilistic learning techniques, if suites of target applications are developed early on.

How should abstraction layers evolve?

- Network stack divisions should not be seen as constants, layers may need to be reshuffled ("telescoped") to allow simpler and more efficient use of hardware and software.
- Network addressing and routing can be application-specific when appropriate, cutting across stack layers, rather than adhering to predetermined boundaries.
- Interfaces and abstractions can be figured out not only during upfront design of software, but by inference from behavior of existing deployed applications. Software can then be automatically adapted during deployment to use new interfaces and primitives.

(e) Risks

What are the risks that threaten the success of XPS research directions?

- Many new research directions, such as introduction of high-level concurrency abstractions, are difficult to integrate seamlessly into existing technologies. Hence, adoption may be hindered by developers being locked into legacy frameworks.
- Widely-used programming languages focus on low-level primitives for concurrency and message passing. This can lead to high-level primitives and frameworks being perceived as outlandish by developers.
- Legacy software cannot be easily adapted to new cloud-based platforms and achieve scalability. Investments made in legacy software often cannot be disregarded from when considering adoption of Platform-as-a-Service solutions.
- Significant effort is being spent on software and programming languages which provide low-level concurrency primitives such as shared variables, threads and locks rather than support concurrency abstractions and high-level concurrency semantics, making rigorous reasoning about runtime behavior of large-scale concurrent systems infeasible.
- A proliferation of similar languages and frameworks that differ in minor details lead to fractured developer communities.
- Non-adoption of formal methods techniques in specification, verification, and testing due to perceived complexities of the formalisms used.
- Inadequate tools and techniques for determining tradeoffs between consistency and availability in distributed systems, and for determining the measure of consistency needed for specific distributed applications.
• Proliferation of proprietary cloud management platforms can threaten availability of transparent quality-of-services guarantees backed by system design. Performant Platform-as-a-Service solutions may require deep integration with hardware not available through Interface-as-a-Service interfaces made available by cloud providers.

How do we guard and hedge against these threats?
• Techniques and tools are needed that focus explicitly on migration of legacy code to efficiently use cloud platforms and leveraging their scaling capabilities.

• Research must demonstrate real-world deployment scenarios, and use of languages and abstractions in settings familiar to developers.

• An important role for research is to consolidate approaches that do not differ in essential aspects, and unify constructs for concurrency and distribution to only those that can be motivated by high-level, application developer friendly abstractions.