Design patterns for programming the fog

M.Danelutto, G.Ferrari, G.Mencagli,
U.Montanari, L.Semini, M.Torquati

FOG vision

User

- Extremely distributed and heterogeneous system
- Suitable programming framework providing mechanisms to implement FOG applications
- Overall the pair may be used to solve the problem at hand

Programmer

- Extremely distributed and heterogeneous system
- Difficult to program and understand
- Requiring different mechanisms, techniques and policies at different levels
  - To be used to implement hierarchical orchestration
  - Implementing the high level mechanisms envisioned by the user

(user = application designer/programmer)

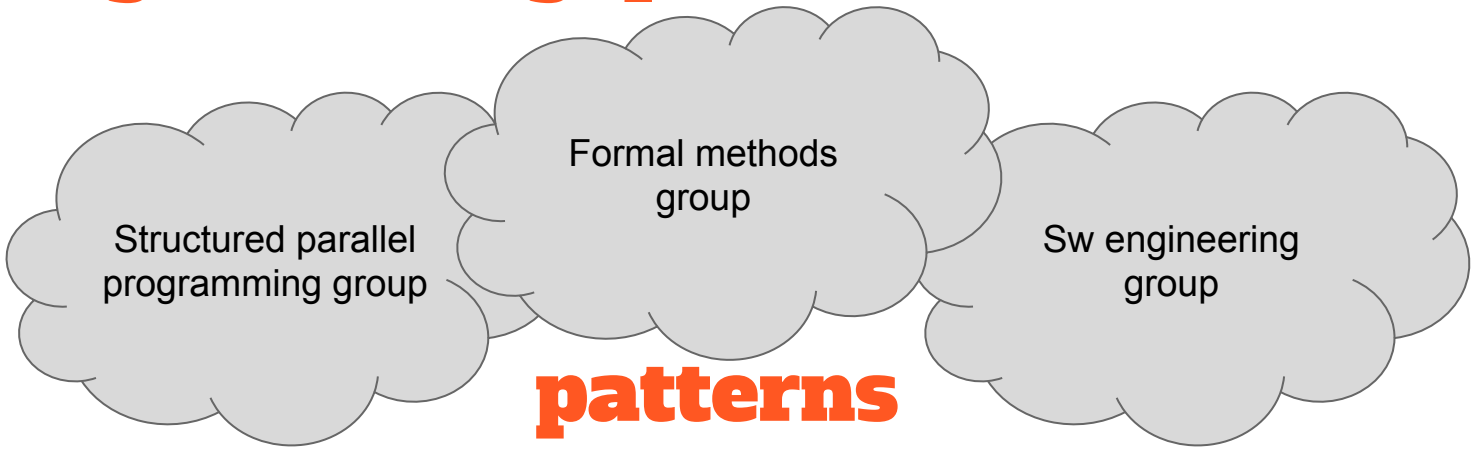
Why patterns with FOG ?

- They decouple algorithm programming issues from implementation issues
- They provide high level of abstraction to the FOG application designer
- The confine FOG complexity in pattern implementation

Overall

- key mechanism to attack FOG app design
- as advocated by different communities
  - HPC (Berkeley report)
  - SwEng
  - Distributed App Programmers

Through The Fog patterns



Synergic FOG app development

Parallel design patterns

Pattern description template

- problem solved
- parallelism exploitation
- different algorithms and policies
- sample implementation code
- typical applications (problems) supported
- ...

Often provided to user (application programmer) through

- ready to use host language mechanisms (classes, libraries, DSL, ...)

Notable examples

- Google MapReduce
- DataFlow graph execution (tensorflow)
- Map, Reduce, Scan, Stencil
- Task farm, Pipeline
- Divide&Conquer
- ...

Key point: separation of concerns

- system programmers implement patterns (optimizations, hw targeting, etc.)
- users only exploit functional semantics of the patterns

Parallel design pattern and RISC-PBB

Any (?) know parallel pattern expressed in terms of terms of RISC-PBB items

- computation patterns (replication, pipeline, tree generation/collapse)
- communication patterns (1-to-n and n-to-1 (multiple policies), feedback (cycles))
- generic composition + data flow semantics

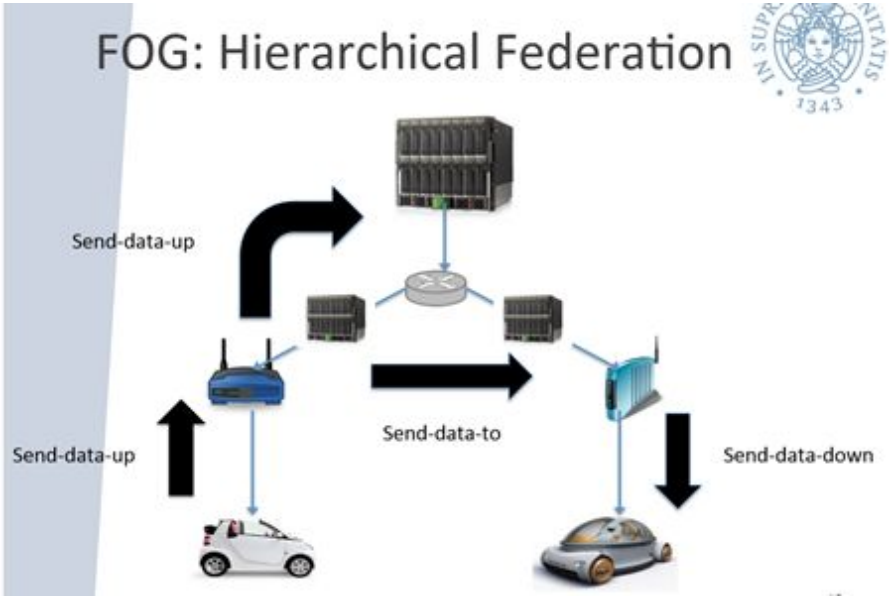
RISC-pbb implemented on top of existing mechanisms

- e.g. FastFlow (threads & shared memory or sockets and COW/NOW)

Formal rewriting rules to introduce optimizations & hardware targeting

Design patterns hierarchy aware

- Context Oriented Programming
- Orchestration/Choreography
- Mob$_{adtl}$ guardians
- Mediator
- Facade
- Chain of Responsibility



Design patterns for hierarchy unaware applications

- Orchestration/Choreography
- Publish-subscribe
- Observer
- P2P

Soft mu-calculus for computational fields

Inspired by semiring mu-calculus

Computation corresponds to fixpoints in a graph-shaped domain

Soft mu calculus formulas ↔ RISC-PBB patterns:

- which adjacent nodes are read
- how their values are combined.

Differerences

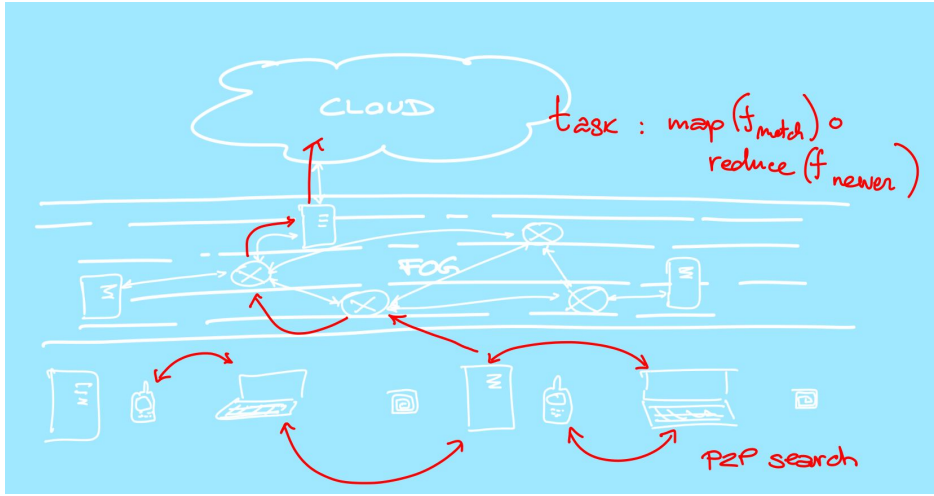
- Smuc: arc labels with a functional meaning;
- RISC-PBB: arcs are connectors to express only flow;
- Smuc: modalities [a] and <a> combine the values received on the arcs;
- formulas: high level meanings, global level;
- formula evaluation semantics: low level communication, evaluation.

Example

- mu calculus for a-reachability
  μZ. init ∨ <a>Z
- soft mu-calculus for shortest distance
  μZ. min(init,<dist>Z)
- fix point approximations: essentially Dijkstra algorithm

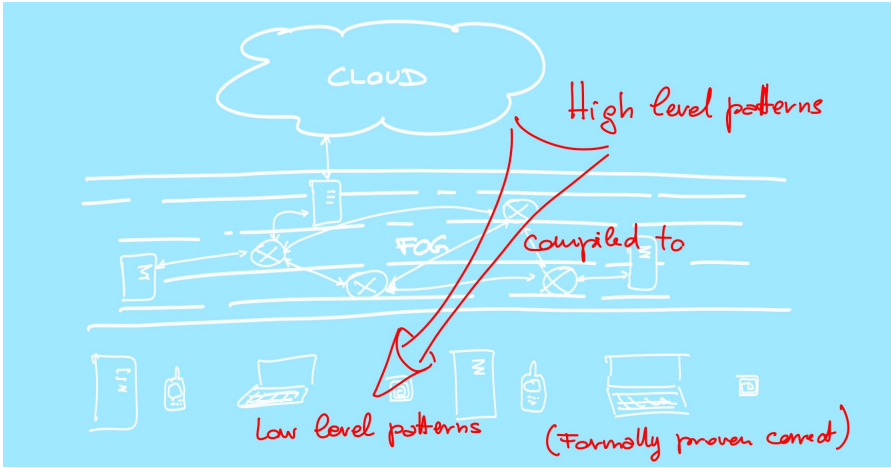
Sample FOG app

Looking up most recent
version of a document



- map (f_{match}) + reduce ($f_{most\ recent}$) pattern
- implemented as
    - low level search: optimising local resources and power, look up documents, deliver the most convenient among :
      <boolean found, data whenmodified, location loc> and <document doc, data whenmodified>
    - high level search and reduce: broadcast search parameters, gather answers, reduce to the most recent document, if not included, retrieve it from loc.
- suitability of mechanisms formally proven (e.g. P2P).
- autonomic management of decisions (e.g. : answer type)

Designing apps (TopDown patterned)



User (application programmer)

- application as proper (comp of) patterns

System programmer

- implements pattern as structured, hierarchical composition of Pattern Building Blocks (connectors and components)
- on top of primitive mechanisms natively supported by FOG components

Feasibility of the implementation

- formalized and demonstrated correct/incorrect through MuCalculus

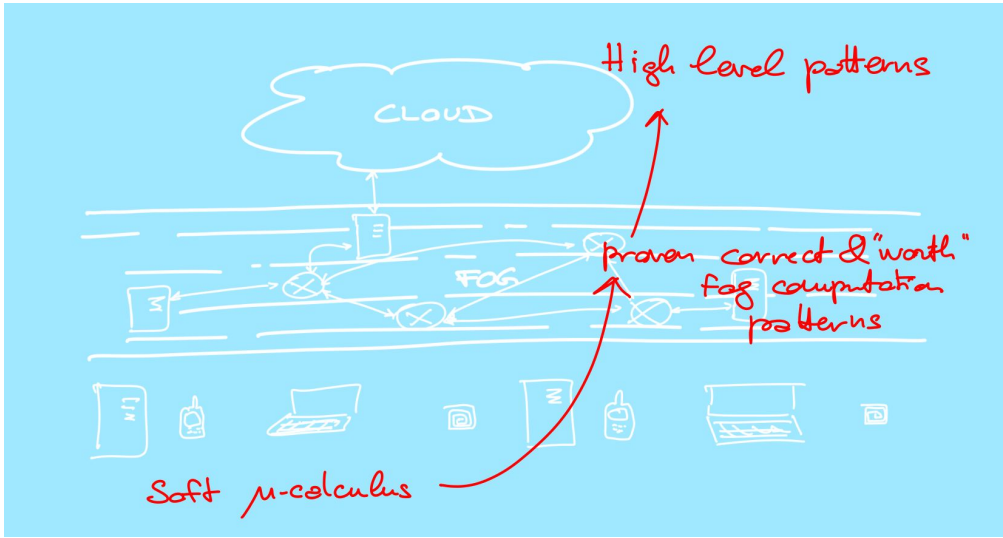
Designing programming frameworks (bottom up, patterned)

FOG infrastructured formalization

- using MuCalculus

Classification of typical computations supported

- provide palette of FOG computation patterns



Provisioning of patterns to end users

- use the palette mechanisms to implement FOG specific patterns

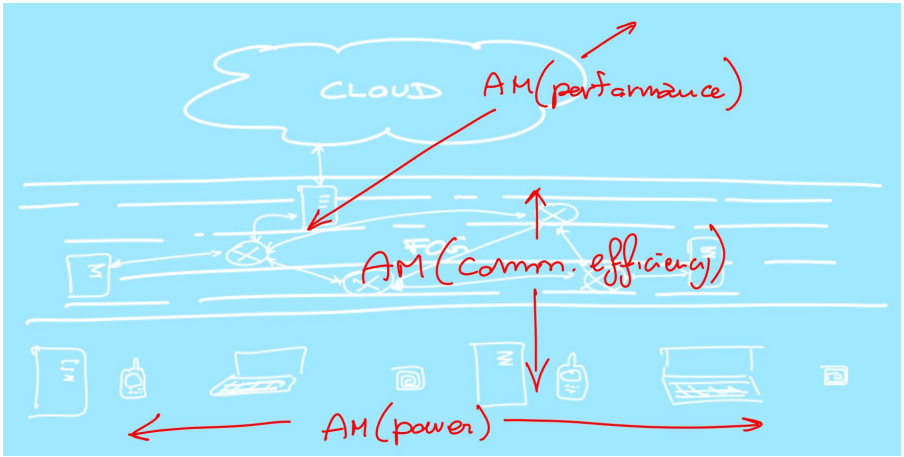
Designing autonomic management

High level management policies

- relative to high level resources and algorithms
- quality of service (user perceived)

Low level management policies

- relative to FOG mechanisms
- quality of service (as perceived/used by tools)



Different policies at different levels

- possibly optimizing different goal functions

Future work

Synergies envisioned

- several distinct directions
- in both an “engineering” and “computer science” first perspective

Research topics individuated

- validation of existing mechanisms
- formal derivation of possible FOG specific mechanisms and computation patterns
- autonomic computing techniques embedded in the tools provided to the FOG programmer

Software engineering perspective

Pattern based application development methodology

- Support Pattern-to-Pattern refactoring techniques to improve FOG app efficiency
- Pattern driven refinement of initial app code through high (pattern) level and low (implementation) level refactoring rules

Derive implementation mechanisms as requirements directly derived from FOG patterns

Introduce autonomic decision levels in pattern implementation and management