A unique confluence of factors have driven the need for cloud computing

• **Demand Pulls**: Process and store large data volumes
  - Y02 22-EB : Y06 161-EB : Y10 988-EB ~ 1 ZB

• **Technology Pushes**: Falling hardware costs & better networks

• **Result**: Aggregation to scale
Since the cloud is not monolithic it is easier to cope with flux and evolve

- Replacement and upgrades are two sides of the same coin

- Desktop: 4 GB RAM, 400 GB Disk, 50 GFLOPS & 4 cores

- 250 x Desktop = 1 TB RAM, 100 TB Disk, 6.25TFLOP and 1000 cores
Cloud and traditional HPC systems have some fundamental differences

- One job at a time = Underutilization
  - Execution pipelines
  - IO Bound activities
- An application is the **sum of its parts**
- Cloud strategy is to **interleave** 1000s of tasks on the same resource
Projects utilizing NaradaBrokering

QuakeSim
Ecological Monitoring: PISCES

Sensor Type
- Dendrometer
- Bubble Flow
- Groundwater Monitor
- Soil Moisture
- Weather
- ET Monitor
- Bannockburn Plantation
- Intensive Study Watershed

Map showing sensor locations near Pawley's Island, S.C. with a close-up of a groundwater monitoring station at Bannockburn Research Site.
Projects utilizing NaradaBrokering

Sensor Grids

RFID Tag    RFID Reader
GPS         Nokia N800
Lessons learned from multi-disciplinary settings

- **Framework** for processing streaming data
- Compute demands will **outpace** availability
- **Manage** computational load transparently
Big Picture

Networked Devices
{Sensors, Instruments}

Simulations & Services

Experimental Models

Burgeoning data volumes
Fueled the need for a new type of computational task

- Operate on dynamic and voluminous data.
- Comparably smaller CPU-bound times
  - Milliseconds to minutes
- **BUT** concurrently interleave 1000s of these long running tasks
- Granules provisions this
Granules is dispersed over, and permeates, distributed components.
An application is the sum of its computational tasks that are ... 

• Agnostic about the resources that they execute on
• Responsible for processing a subset of the data
  – Fragment of a stream
  – Subset of files
  – Portions of a database
Granules does most of the work for the applications except for ...

1. Processing Functionality
2. Specifying the Datasets
3. Scheduling strategy for constituent tasks
Granules processing functionality is domain specific

• Implement just one method: `execute()`
• Processing 1 TB of data over 100 machines is done in 150 lines of Java code.
Computational tasks often need to cope with multiple datasets

- **TYPES**: Streams, Files, Databases & URIs
- **INITIALIZE**: Configuration & allocations
- **ACCESS**: Permissions and authorizations
- **DISPOSE**: Reclaim allocated resources
Computational tasks specify their lifetime and scheduling strategy

- **Permute** on any of these dimensions
- **Change** during execution
- **Assert** completion

Data availability

Number of times

Periodicity
Granules discovers resources to deploy computational tasks

- **Deploy** computation instance on multiple resources
- **Instantiate** computational tasks & execute in Sandbox
- **Initialize** task’s **STATE** and **DATASETS**
- **Interleave** multiple computations on a given machine
Deploying computational tasks

Content Distribution Network
Granules manages the state transitions for computational tasks

**Transition Triggers:**
- Data Availability
- Periodicity
- Termination conditions
- External Requests
Activation and processing of computational tasks

Dispose

Execute

Dormant
MAP-REDUCE enables concurrent processing of large datasets
Substantial benefits can be accrued in a streaming version of MAP-REDUCE

• File-based = Disk IO  →  Expensive
• Streaming is much faster
  – Allows access to intermediate results
  – Enables time bound responses
• Granules Map-Reduce based on streams
In Granules MAP and REDUCE are two roles of a computational task

- **Linking** of MAP-REDUCE roles is easy
  - `M1.addReduce(R1)` or `R1.addMap(M1)`
  - Unlinking is easy too: `remove`

- Maps **generate** result streams, which are consumed by reducers

- Reducers can **track** outputs from Maps
In Granules MAP-REDUCE roles are interchangeable

```java
RM.addReduce(M1)
RM.addReduce(M2)
RM.addReduce(M3)
RM.addReduce(M4)
```
Scientific applications can harness MAP-REDUCE variants in Granules

- **ITERATIVE**: Fixed number of times
- **RECURSIVE**: Till termination condition
- **PERIODIC
- **DATA AVAILABILITY DRIVEN**
Complex computational pipelines can be set up using Granules

- Iterative, Periodic, Recursive & Data driven
- Each stage could comprise computations dispersed on multiple machines
Granules manages pipeline communications complexity

- No arduous management of **fan-ins**
- Facilities to **track** outputs
- **Confirm** receipt from all preceding stages.
Granules allows computational tasks to be cloned

- Fine tune redundancies
- Double-check results
- Discard duplicates from clones
Related work

- **Hadoop**: File-based, Java, HDFS
- **Dryad**: Dataflow graphs, C#, LinQ, MSD
- **Disco**: File-based, Erlang
- **Pheonix**: Multicore
- **Google Cloud**: GFS
Granules outperforms Hadoop & Dryad in a traditional IR benchmark

The graph shows the overhead for histogramming words (in seconds) plotted against the total size of the dataset (in GB). The overhead increases linearly with the size of the dataset for all systems, with Granules consistently showing the lowest overhead across all dataset sizes.
Clustering data points using K-means

![Graph showing the overhead (seconds) vs. number of 2D data points (millions) for different methods.](image)

- **Hadoop**
- **Granules**
- **MPI**

- Overhead (Seconds)
- Number of 2D data points (millions)
Computing the product of two 16Kx16K matrices using streaming datasets

Each Granules instance deals with at least 18K streams

- Number of Machines
- Computing overhead in Seconds
Maximizing core utilizations when assembling mRNA sequences

Program aims to reconstruct full-length mRNA sequences for each expressed gene
Preserving scheduling periodicity for $10^4$ concurrent computational tasks
The streaming substrate provides consistent high performance throughput.
The streaming substrate provides consistent high performance throughput.
Cautionary Tale: Gains when Disk-IO cannot keep pace with processing
Key innovations within Granules

- **Easy** to develop applications
- Support for real-time **streaming datasets**
- Rich **lifecycle** and scheduling support for computational tasks.
- Enforces semantics of complex, distributed computational **graphs**
- Seamless **cloning** at finer & coarser levels
Future Work

• **Probabilistic** guarantees within the cloud
• Efficient generation of compute streams
• **Throttling** and steering of computations
• **Staging datasets** to maximize throughput
• Support **policies** with global & local scope
Conclusions

• Pressing \textbf{need} to cloud-enable network data intensive systems

• \textbf{Complexity} should be managed \textbf{BY} the runtime, and \textbf{NOT} by domain specialists

• \textbf{Autonomy} of Granules instances allows it to cope well with resource pool expansion

• Provisioning lifecycle metrics for the \textbf{parts} makes it easier to do so for the \textbf{sum}