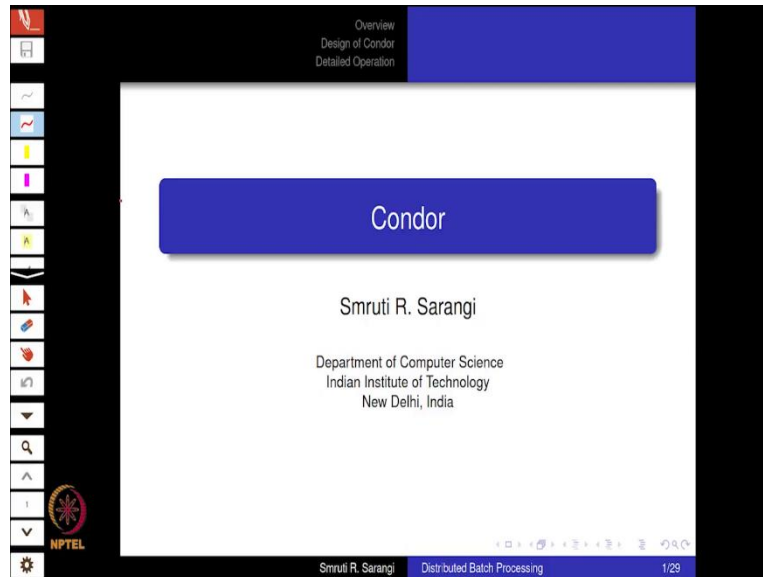


**Advanced Distributed Systems**  
**Professor Smruti R. Sarangi**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Delhi**  
**Lecture 24**  
**Condor: Distributed Batch Processing System**

(Refer Slide Time: 0:18)



Slide 1: Condor

Overview  
Design of Condor  
Detailed Operation

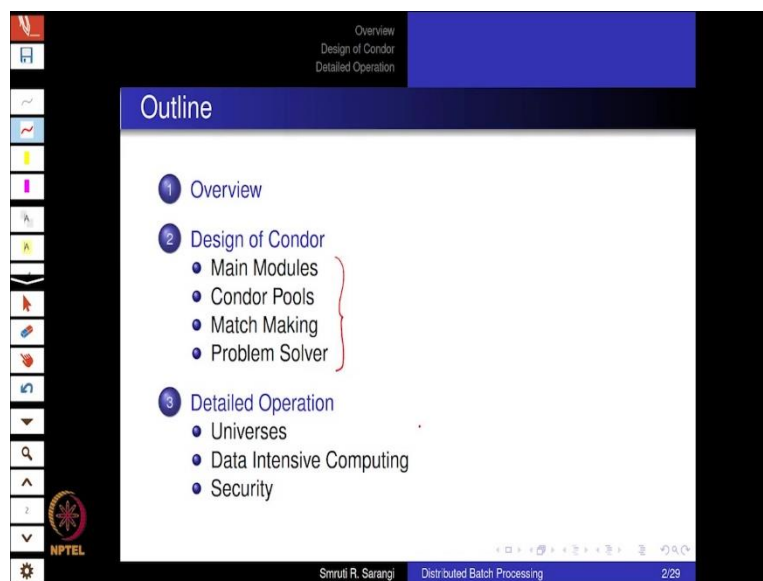
**Condor**

Smruti R. Sarangi

Department of Computer Science  
Indian Institute of Technology  
New Delhi, India

NPTEL

Smruti R. Sarangi Distributed Batch Processing 1/29



Slide 2: Outline

Overview  
Design of Condor  
Detailed Operation

**Outline**

- 1 Overview
- 2 Design of Condor
  - Main Modules
  - Condor Pools
  - Match Making
  - Problem Solver
- 3 Detailed Operation
  - Universes
  - Data Intensive Computing
  - Security

NPTEL

Smruti R. Sarangi Distributed Batch Processing 2/29

In this lecture we will discuss the Condor distributed job processing system. So, we will first go through an overview of what a distributed job management or a distributed batch processing system is and then we will discuss the main modules of Condor and the detailed operation.

(Refer Slide Time: 0:40)

The slide is titled "History of Condor" and is part of a presentation on "Distributed Batch Processing". It contains a bulleted list of points with handwritten annotations in red and blue ink. The points are:

- Towards the mid 80s, the power of distributed computing was realized
- Clusters of machines could outperform supercomputers (with "(desktops)" written above "clusters")
- There was a need for a middleware to integrate third party computers
  - Integrate computers with different types of hardware and software
  - Provide consistency and reliability guarantees
  - Provide security, and trust (with "job" written next to it)
  - Ensure fairness among users
  - Be able to efficiently run large scale distributed jobs.

At the bottom of the slide, a pink box contains the text: "Condor was thus born in the University of Wisconsin". The slide footer includes "Snruti R. Sarangi" and "3/29".

So, the idea was that towards the mid-80s, the power of distributed computing was realized. So, it was realized that a single machine, regardless of the size of the machine, be it a small machine like a desktop computer or a large machine like a mainframe, or even a supercomputer. That is not sufficient for most problems. And also, even if it is sufficient for a very restricted set of problems, and it is not very flexible.

So, the idea was to create a cluster of machines where, so these could be normal regular machines in the lab aggregate their computing power. Such that they could outperform supercomputers. So, that was the idea. So, what led to this is that by the end of the late 80s, and early 90s, many households, particularly in the US and Europe started to have desktop computers. So, these computers were not used at nighttime.

So, the idea was that, why not, we take over these computing resources when they are not being used, and kind of have a large distributed system of all of these comprising all of these small desktop machines that each of the individual users had. So, essentially a cluster of desktops that did not everybody had, which people had in their homes, and then a take a large piece of work divided into small chunks, and distribute these small chunks of work across these machines.

So, this did sound to be a very interesting idea. The reason being that there was a huge amount of compute capacity that was not being utilized. And this could be utilized for problems that did not require the kind of synchronization, which a typical, job on a supercomputer would require. And that would be something like inverting a matrix or solving a large differential

equation. So, for those supercomputers are better. But for other jobs, where, for example, we need to run a large simulation with a large number of parameters.

And if they and let us say that we want to run just 100 copies of the same simulation with different parameters, we can distribute one copy on each desktop. So, there was a need for a middleware. So, middleware is kind of like a runtime system over the operating system to integrate all of these third-party computers.

So, idea was to integrate computers with different types of hardware and software, provide consistency and reliability guarantees. So, why consistency? Because we need to have some idea of how these jobs execute. So, they may not be fully serializable. But at least, there should be some rules to the game.

And second, we need some reliability in the sense, we should be able to trust the results that we get in terms of their correctness, as well as the security and trust aspect, where essentially, if a job is running on a remote desktop, it should not be possible for the job to hack into the desktop. And it should not be possible for the desktop to tamper with the job.

So, these are third party jobs. So, pretty much let us say that today if I put my desktop on this desktop cloud. So, this thing has a new name, also is called the desktop as a service. But let us say it is a compute cloud. So, if I like to put my desktop on a compute cloud, then the idea is that jobs then can run on my desktop. And but the thing is that I will not know which job is running on my desktop.

So, it is possible that it might try to corrupt my system or the converse is also possible where I have a malicious intent, and I am trying to corrupt that job. So, for both the cases security and trust are required. Additionally, if we have a large distributed system of machines, we need some notion of fairness. Otherwise, we will not be able to guarantee the timely completion of these jobs. And second, it should be easy to efficiently run such large-scale distributed jobs.

So, the first such system to actually propose this was the Condor project, which initially was born in the University of Wisconsin, and later on, as it kind of grew. So, now, condor is like an open-source project. So, this is something that anybody can download and run. So, as long as there is a cluster of machines, we can install Condor on all of them. And to any outside user, they would actually appear, the Cluster would actually appear to be a single machine.

(Refer Slide Time: 5:55)

The slide is titled "Philosophy of Condor" and features a hand-drawn diagram of a cluster machine labeled "SeTI". The diagram shows a central oval with four rectangular nodes attached to its sides. The text "SeTI" is written to the right of the diagram. Below the diagram is a numbered list of five principles:

- 1 Flexibility
- 2 Let communities grow naturally – Build software that permits co-operation among users.
- 3 Leave the owner of the computing resource in control.
- 4 Make the system fault tolerant
- 5 Lend and borrow from other disciplines.

The slide also includes a navigation bar at the top with "Overview", "Design of Condor", and "Detailed Operation". The NPTEL logo is visible in the bottom left corner, and the footer contains "Snruti R. Sarangi", "Distributed Batch Processing", and "4/23".

So, what are the advantages of Condor? Well, the main advantage is flexibility. Because we can take a lot of desktop power that we are getting. And then we can sort of integrate all of it into one single large cluster machine. So, much of this also started with the SeTI project. So, the SeTI project was a project to find out if extra-terrestrial life exists. So, for this, a lot of information that was actually captured by telescopes had to be analyzed to find traces of life outside Earth.

So, the SeTI project required the combination capacity that it needed. That was not available formally. So, that is the reason these desktops were taken over, where people pretty much contributed compute power to a cloud. And the cloud then ran jobs on them. So, what did users get in return? Well, they got some money. So, in all such cases, these need not be fully philanthropic altruistic activities. If I am contributing compute power, I get some money in return. And that kind of allows me to sort of invest in more compute power and contribute.

So, in this case, communities grow naturally. And so, they build a bond of trust with other communities such that we have a large cluster of machines with a large amount of compute power, and this compute power can be used to support the needs of a very large community where each job can be very different and the jobs will be heterogeneous, but on this large set of machines, it would be possible to run them.

So, condor has some basic principles, the first is that we leave the owner of the computing resource in control. Otherwise, owners would not come forward to actually donate their machines to the pond or cloud. So, it should be possible for an owner for example, to terminate

the remote job and do work of his own, the system should be fault tolerant, which means that if one machine crashes another machine can take over.

And also, it would be possible to lend and borrow concepts in this in the design of Condor. So, many concepts have been lent and borrowed from other disciplines, particularly from parallel computing, distributed computing, to actually create the system and also, traditional things like Semantic web as we will see.

(Refer Slide Time: 8:46)

Overview  
Design of Condor  
Detailed Operation

## Condor High Throughput Computing System

- Condor provides a method for a set of users to submit their jobs in batch mode.
- Condor provides:
  - Job Management Mechanisms ←
  - Scheduling Policies ←
  - Resource Monitoring
  - Resource Management

• Planning  
• Scheduling

virtual pool

NPTEL

Srnul R. Sarangi Distributed Batch Processing 5/29

So, Condor provides a method for a set of users to of course, submit their jobs. So, this is done in batch mode. These are not interactive jobs, they are batch jobs, which means that they run and whenever they are done, the user gets an email that the job is done. So, Condor provides job management mechanisms. This means that it provides methods to manage a running job.

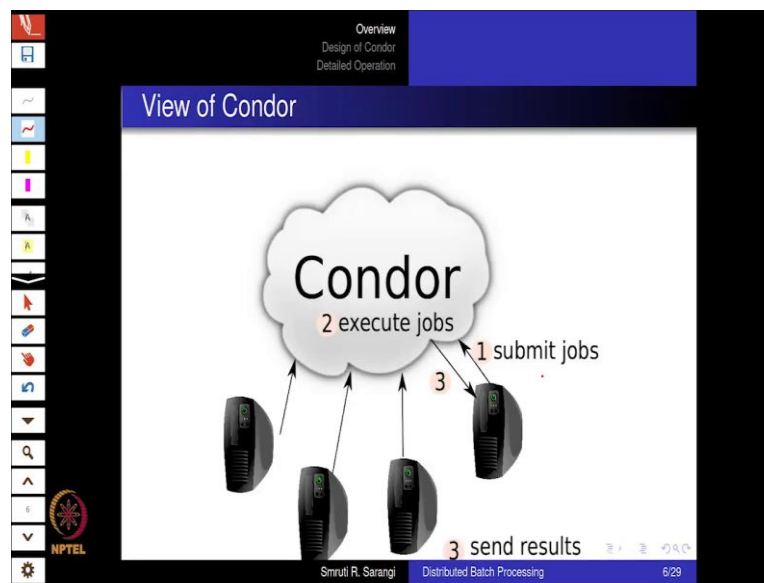
Scheduling policies, which tell the user how the job should actually be run. Resource monitoring, which is to monitor the resources, how much they are being used, and finally manage the resources. So, Condor is not really one large pool, but it is actually a collection of several small pools or machines.

So, we can have one small pool of let us say the IIT Delhi machines, another pool of another machines. So, we can aggregate all of these and make one large virtual pool of machines. So, in this there are two important concepts that I would like to mention one is called planning and the other is scheduling.

So, planning is when we take a large job. So, let us say a large job requires 200 machines. And we decide that we will take 100 machines from here, 50 machines from here and 50 machines from here. So, this is broadly planning.

Scheduling is the next phase for each of these individual clusters, decides when and how to run these jobs, because they will have local jobs as well. So, when and how to run these jobs such that a given metric is maximized or minimized.

(Refer Slide Time: 10:42)



So, this is a broad idea that we have a large cloud of compute nodes called the Condor cloud. And in each one of them, we have individual users who submit jobs, the cloud executes jobs, and then finally, the results are given back to the individual machines.

(Refer Slide Time: 11:05)

Outline

- 1 Overview
- 2 Design of Condor
  - Main Modules
  - Condor Pools
  - Match Making
  - Problem Solver
- 3 Detailed Operation
  - Universes
  - Data Intensive Computing
  - Security

Smruti R. Sarangi | Distributed Batch Processing | 7/29

Main Modules in Condor

- **ClassAds System** : This is a language that lets users specify the type of the job, the type of the resource offered to the cloud, and the matching policies.
- **Execution Engine** : Executes users' jobs (respects DAG based constraints) on a large grid.
- **Job Checkpoint and Migration** Can transparently checkpoint jobs and can migrate them among machines. For example, if a user on an idle desktop presses a key, then any Condor job running on it seamlessly migrates to another machine.
- **Remote Sandbox** : All I/O related system calls are redirected to the machine that submitted the job.

Smruti R. Sarangi | Distributed Batch Processing | 8/29

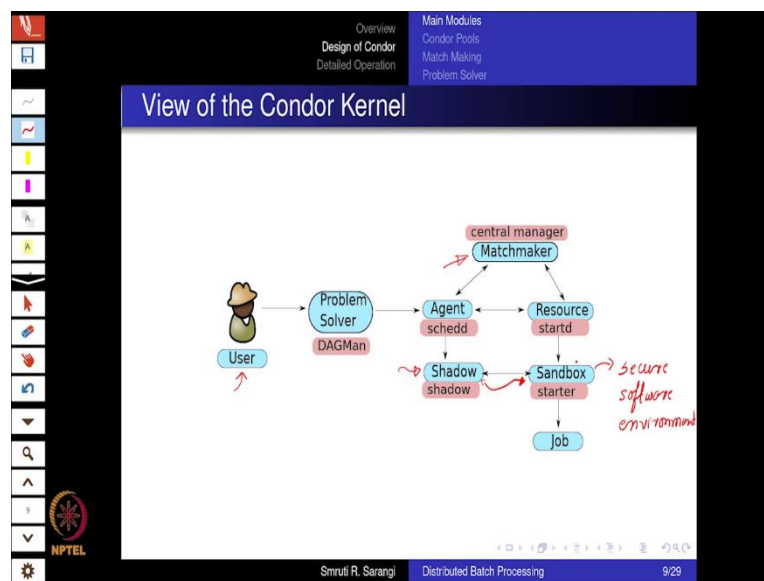
So, now, we will discuss the main modules of Condor. So, the main modules in a condor system are as follows. So, we have a ClassAds system. So, this is a language that lets users specify the type of the job. So, this can include the type of the machine that is required the operating system, the kind of software environment that is required, and the type of the resource that is offered by the cloud.

So, this is used to specify both the job as well as the resource and any kind of a matching policy because after all jobs have to be matched with resources. Then we have an execution engine, which executes the user's jobs. So, it respects the constraints of it of a job. So, job may not be a single program, but it can be a set of programs, where the output of one program feeds into the input of the other program.

So, in this case, the execution engine respects all of that, and it runs the jobs on a large grid of machines. So, typically, the constraints are shown are represented using a directed acyclic graph a DAG. Furthermore, a great feature of condor is that it allows job checkpoint and migration. What this basically means is that let us say that on a machine a job is running. After that in the middle of so let us say this the execution time of a job. In the middle of a job, the user presses a key, which means the user is now active, the owner of the desktop is active.

In this case, we checkpoint the state of the job. And we migrate the job to another machine, where it starts at exactly the same point in its execution, and continues to execute. So, the checkpoint and migration is a key feature of Condor, which allows even running jobs to seamlessly migrate between machines. Furthermore, we will define something called a remote sandbox, which kind of dictates how jobs run on remote machines. Well, let us defer this discussion to a later slide.

(Refer Slide Time: 13:29)



So, the structure of the Condor system, the Condor kernel is like this, we have the user. Then we have the problem solver. So, the problem solver is essentially a software module, which takes the jobs, the structure of the jobs that are represented as a directed acyclic graph and for each job. So, the job is given to a condor agent. So, the agent registers itself with a matchmaker, which tries to pair it with a resource.

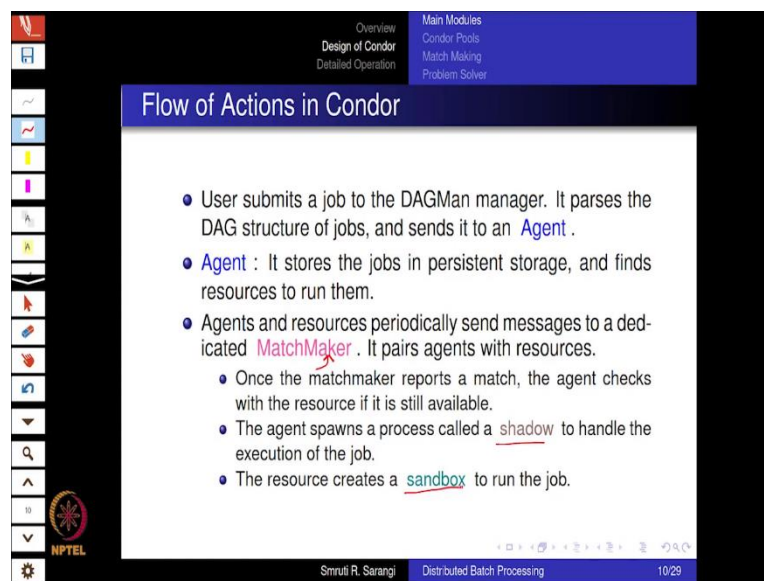
So, the role of the matchmaker share is to pair a job with a resource. And once a job gets the kind of resource that it requires from the Condor system with the help of the matchmaker. So, after that, what happens is that the agent starts a process called shadow. So, the job of the



shadow is to help the sandbox on the resource run the job. So, we will discuss this in detail later, how the shadow and the sandbox actually talk to each other how they collaborate.

And then the job runs on the sandbox on the resource. So, we can think of the sandbox as a secure software environment that runs the job. So, that is the main aim, the Sandbox is a secure software environment to run the job. Well, the job cannot maliciously access the resources, the access features of the resource, and the resource cannot harm the job. So, it is a two-way sandbox.

(Refer Slide Time: 15:31)



The slide is titled "Flow of Actions in Condor" and is part of a presentation on "Design of Condor". The slide content is as follows:

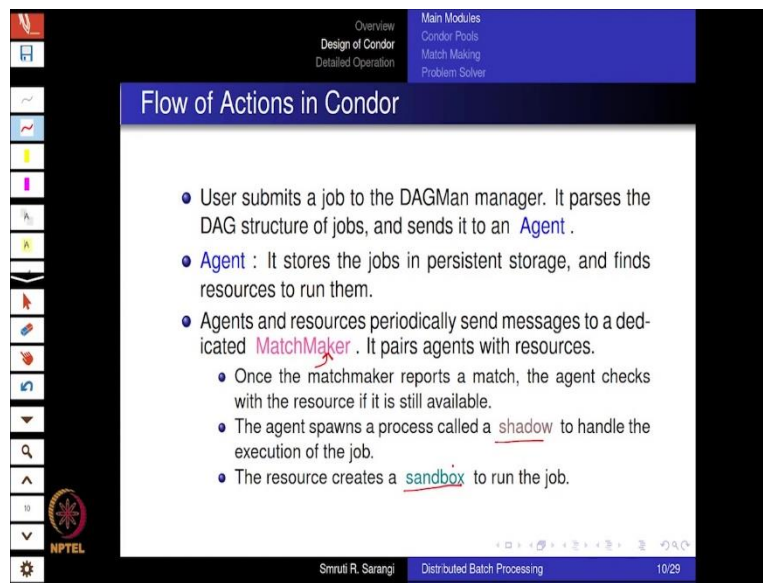
- User submits a job to the DAGMan manager. It parses the DAG structure of jobs, and sends it to an **Agent**.
- **Agent** : It stores the jobs in persistent storage, and finds resources to run them.
- Agents and resources periodically send messages to a dedicated **MatchMaker**. It pairs agents with resources.
  - Once the matchmaker reports a match, the agent checks with the resource if it is still available.
  - The agent spawns a process called a **shadow** to handle the execution of the job.
  - The resource creates a **sandbox** to run the job.

The slide also features a navigation sidebar on the left, a top menu with "Overview", "Design of Condor", and "Detailed Operation", and a bottom footer with "Snruti R. Sarangi", "Distributed Batch Processing", and "10/23".

So, the flow of actions in condor is like this, the user submits a job to the DAGMan manager. It parses the DAG structure of jobs. And sends, then a job one after the other to an agent. The agent stores the jobs in persistent storage and find resources to run them. Agents and resources periodically send messages to a dedicated matchmaker whose job is to essentially pair jobs and resources it pairs agents with resources.

Once the matchmaker reports a match, the agent checks with a resource if it is still available. Because it is possible that the resource might not be available because the local user on it is using it. So, then the agent spawns a process called a shadow to handle the execution of the job. The resource creates a sandbox to run the job.

(Refer Slide Time: 16:35)



Overview  
Design of Condor  
Detailed Operation

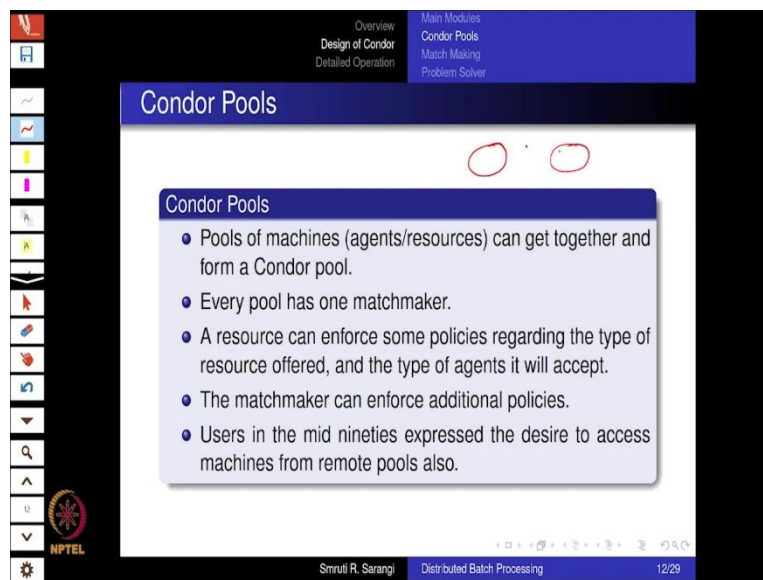
Main Modules  
Condor Pools  
Match Making  
Problem Solver

### Flow of Actions in Condor

- User submits a job to the DAGMan manager. It parses the DAG structure of jobs, and sends it to an **Agent**.
- **Agent** : It stores the jobs in persistent storage, and finds resources to run them.
- Agents and resources periodically send messages to a dedicated **MatchMaker**. It pairs agents with resources.
  - Once the matchmaker reports a match, the agent checks with the resource if it is still available.
  - The agent spawns a process called a **shadow** to handle the execution of the job.
  - The resource creates a **sandbox** to run the job.

NPTEL

Smruti R. Sarangi Distributed Batch Processing 10/29



Overview  
Design of Condor  
Detailed Operation

Main Modules  
Condor Pools  
Match Making  
Problem Solver

### Condor Pools

- Pools of machines (agents/resources) can get together and form a Condor pool.
- Every pool has one matchmaker.
- A resource can enforce some policies regarding the type of resource offered, and the type of agents it will accept.
- The matchmaker can enforce additional policies.
- Users in the mid nineties expressed the desire to access machines from remote pools also.

NPTEL

Smruti R. Sarangi Distributed Batch Processing 12/29

So, we will not discuss Condor pools. So, pools are machines, pools are either agents or resources can get together and form a condor pool. So, Condor pool is a pool of agents or of resources. Every pool has a matchmaker. So, resource can enforce some policies regarding what kind of resources offered and the matchmaker can have additional policies. And so, the default is of course, that we get a machine in the same pool. But in the mid-90s, Condor started expanding this to get machines from remote pools as well.

(Refer Slide Time: 17:21)

The image is a screenshot of a presentation slide. At the top, there is a navigation menu with 'Overview' and 'Main Modules' (Condor Pools, Match Making, Problem Solver). The slide title is 'Gateway Flocking'. Below the title, there is a diagram showing three interconnected nodes labeled G1, G2, and G3. The main content is divided into two sections: 'Gateway Flocking' and 'Direct Flocking'. The 'Gateway Flocking' section contains three bullet points: 'Every pool will have a gateway that can interact with gateways of other remote pools.', 'If a pool has an idle machine, then its gateway can send its advertisement to other gateways.', and 'They can forward this information in their local pools.' The 'Direct Flocking' section contains one bullet point: 'An agent reports itself to multiple matchmakers, and effectively joins multiple pools.' The slide footer includes the NPTEL logo, the name 'Srnulli R. Sarangi', the course 'Distributed Batch Processing', and the slide number '13/29'.

So, there were two approaches gateway flocking and direct flocking, which was to get resources from remote pool. So, in gateway flocking, every pool will have a gateway that can interact with gateways of other remote pools. So, if this is a pool this will have a gateway that can interact with the gateway of other remote pools.

And if let us say in this pool, there is an idle machine. Then what can happen is a gateway can let other gateways know about idle machine. So, if there is a request from within the pool, this can be forwarded to the other pool that has an idle machine. And then the idle machine can be assigned to one of the jobs in this pool. So, this way, we can forward information among pools.

Direct flocking is when an agent reports itself to multiple matchmakers. It does not actually have to go through a gateway. So, it can directly report itself to multiple matchmakers in different pools, which means it effectively joins multiple pools and it gets resources from them.

(Refer Slide Time: 19:00)

The slide is titled "Interaction with Globus" and is part of a presentation on Condor. It features a navigation menu on the left and top. The main content is a list of bullet points. Handwritten red notes are present: "direct flocking" and "gateway flocking" are written above a red box labeled "pool". A red line connects the text "Condor-G." in the third bullet point to the "pool" box.

Overview  
Design of Condor  
Detailed Operation

Main Modules  
Condor Pools  
Match Making  
Problem Solver

### Interaction with Globus

- Direct and gateway flocking are complicated.
- In the late nineties, the Globus toolkit emerged:
  - It was a standard architecture to interconnect clusters and grids.
  - Provided trust, security, and secure file transfer services.
  - **GRAM** Protocol: Grid Resource Access and Management
  - Condor interacts with **GRAM** using a dedicated module called Condor-G.

direct flocking  
gateway flocking  
pool

NPTEL  
Smruti R. Sarangi Distributed Batch Processing 14/29

So, direct flocking and gateway flocking used to be the main approaches used to be the mainstay of Condor till quite some time. Then in the late nineties, the Globus toolkit for managing grids of computers emerged. So, it was a very standard architecture to interconnect clusters and grids. It provided trust security and secure file access and transfer services using the gram protocol, the grid resource access and management protocol.

So, then, along with direct flocking, which is when a agent kind of registers with multiple matchmakers in different pools, and gateway flocking where each tool has its gateway. So, essentially agent contacts its local gateway and then the local gateway contacts remote gateways. A new an extension to Condor was added where the underlying architecture was actually the Globus cloud architecture.

And the Globus cloud kind of took care, kind of virtualized the entire cloud system for Condor, such that Condor saw the entire cloud as one large cluster of machines, and the notion of pools kind of went away, because the entire cloud became a single pool.

(Refer Slide Time: 20:26)

The slide shows a navigation menu with the following items:

- Overview
- Design of Condor
  - Main Modules
  - Condor Pools
  - Match Making
  - Problem Solver
- Detailed Operation
  - Universes
  - Data Intensive Computing
  - Security

At the bottom of the slide, it says "Smruti R. Sarangi" and "Distributed Batch Processing 15/29".

The slide is titled "Match Making" and contains an "Overview of Match Making" box with the following steps:

- Agents and resources advertise their details using small snippets of text called ClassAds.
- The matchmaker pairs agents and resources.
- The agent then goes and **claims** the resource.

Handwritten in red below the list is the word "availability".

At the bottom of the slide, it says "Smruti R. Sarangi" and "Distributed Batch Processing 16/29".

So, with these three flocking approaches in mind, let us now come to matchmaking in Condor. So, the matchmaking is that both agents as well as resources, advertise their details using small snippets of text called Class Ads. So, the main aim is to pair agents and resources based on their Class Ads.

And then once there is a match, the agent goes and claims the resource. So, it is of course possible that in the time being the resources may be dead, the resources may be busy. So, of course, availability needs to be checked. And if the resource is not available, then the matchmaker needs to be contacted once again.

(Refer Slide Time: 21:19)

```
Job ClassAd
[
  MyType = "Job"
  TargetType = "Machine"
  Requirements = ((other.Arch == "INTEL" && other.OPSy
  Rank = (Memory * 10000) + KFlops
  Cmd = "abc/abc.exe"
  Owner = "myself"
]
```

- "Requirements" indicates the constraints
- "Rank" is the objective function of the match
- Among the available resources, the matchmaker chooses the highest rank

Smruti R. Sarangi | Distributed Batch Processing | 17/29

So, a typical Class Ads would look something like this. So, my type, the type of the ad, so this is a job, it could be resource as well. The target type, the machine is the requirements of the machine are like this, the other operating system should be Linux, let us say, the rank is equal to the amount of memory we need. So, we will discuss the rank in some detail. So, the rank is pretty much a function that is used to evaluate the suitability of a resource.

So, of course, the rank can be defined in different ways. So, the way that this Class Ads is defining the rank of a machine for a job is memory times 10,000 plus the KFlops. So, flop is number of floating-point operations per second. So, this is essentially specifying a function of how a job would assess a resource. So, then, so these are all the requirements of what kind of a machine we need to run on.

And if there are multiple machines that match the requirements, then of course, we use the rank function and find the suitability of machines. And of course, we will choose the machine that is the most suitable. The command is the command that is that will be executed on the remote machine. So, in this case, it is abc dot exe directory, abc, and the owner so owner is, who is the owner of the job.

So, owner of the job is myself, this is essentially my user ID. So, as mentioned requirements indicate the constraints. The rank is like an objective function of the match that we can use to assess how well the match is. And among the available resources, of course, a matrix the matchmaker chooses the resource with the highest rank.

(Refer Slide Time: 23:27)

The screenshot shows a presentation slide with a dark blue header and a white content area. The header contains navigation links: 'Overview', 'Design of Condor', 'Detailed Operation', 'Main Modules', 'Condor Pools', 'Match Making', and 'Problem Solver'. The slide title is 'Enhancements to Matchmaking'. The content area lists five bullet points, each with a colored dot and underlined text:

- Support for writing custom Java and C modules
- Gang matching – coallocation of more than one resource (machine and license)
- Collections provide database support for saving ClassAds
- Set matching involves selecting a large number of classads
- Named references permit one classAd to refer to another one.

Below the text, there is a hand-drawn diagram in red showing two boxes connected by an arrow pointing from the left box to the right box. The slide footer includes the NPTEL logo, the name 'Snruti R. Sarangi', the title 'Distributed Batch Processing', and the page number '18/23'.

So, further enhancements, future enhancements were made to matchmaking. Support was added for writing custom Java and C modules. For even more sophisticated matchmaking. Also, Gang matching was allowed. Which means that so many a times, we might want to run a program like MATLAB, that comes with a license on a given machine. So, in this case, we will like to co-allocate the MATLAB license and the machine.

So, this kind of coallocation of more than one resource was is known as gang matching. So, suppose this was added because many times in a cluster, we like to run jobs that use software with certain licensing requirements. Or it is possible that certain licenses are available on certain machines, and they are not available on certain machines. So, gang matching kinds of allows that.

Then the support for collections was added, which provides some support for saving Class Ads such that even if the matchmaker crashes, we can retrieve the Class Ads and do a matching once the matchmaker comes up. And then there was set matching where instead of matching a single Class Ads, we can match a set of Class Ads and also named references were added where essentially one Class Ads can refer to another Class Ads which means that if there is some information in this Class Ads, this is automatically being used by the one that it is, the one that is referring to it.



(Refer Slide Time: 25:19)

The slide shows a navigation menu with the following items:

- Overview
- Design of Condor
  - Main Modules
  - Condor Pools
  - Match Making
  - Problem Solver
- Detailed Operation
  - Universes
  - Data Intensive Computing
  - Security

Handwritten annotations in red ink:

- Arrows pointing from 'Condor Pools' to 'direct', 'gateway', and 'Globus'.
- An arrow pointing from 'Problem Solver' to the right.

Slide footer: Smruti R. Sarangi, Distributed Batch Processing, 19/29

So, now, what, how far have we gone? Well, we have discussed the broad philosophy of Condor, we have discussed Condor pools. So, in this we have discussed direct flocking, gateway flocking and we have discussed running Condor on a great with its own middleware, which was essentially running Condor on the Globus toolkit. Now, we will discuss the problem solver, which is essentially the core execution part of condor.

(Refer Slide Time: 25:55)

The slide is titled "Problem Solver – Master-Worker Mode" and contains the following text:

- The Master-Worker mode has one master process that directs the work of many worker processes
- The master contains
  - work-list** : Record of all the outstanding work that needs to be performed
  - tracking-module** : Keeps track of remote processes, and allots them work items.
  - steering-module** : Examines the results of workers, modifies the-work lists, and co-ordinates with Condor
- Workers can die at any time. The tracking module then returns them to the work list. *Side-effect free*
- The tracking module can replicate work items (work item should not have side effects)

Handwritten annotations in red ink:

- A diagram showing a box labeled 'M' (Master) connected to three boxes labeled 'W' (Workers).
- A box labeled 'M' with a red 'X' over it, with an arrow pointing to the text "Workers can die at any time..."

Slide footer: Smruti R. Sarangi, Distributed Batch Processing, 20/29

So, condor has a master worker mode, where there is one master process that directs the work of many worker processes. So, the master process on each machine has a work list think of this as a work queue. So, this maintains a record of all outstanding work all the jobs that need to be



performed. Then, we have a tracking module which keeps track of all the remote processes and it alerts them work items.

So, the weights the master worker model of condor is essentially there is one Condor master server. And there are like many of these worker kind of slave servers. So, in this master worker model or this master slave model, so, I should rather maybe use the master worker model, that is what the people refers to. So, the master maintains a work list a work queue of what work needs to be done.

And within this work list, these are assigned to different workers and a tracking module tracks, how far the workers are progressed. And a steering module examines the results generated by the workers, modifies the workplace and coordinates with condor. So, here the idea is that workers can of course die at any time, they can crash at any time, the tracking module then returns the kind of undone work back to the work list.

So, I start the work can be assigned to another worker. So, of course, in such cases, we assume that the job is side effect free, which means that it has not access to other resources. For example, if a job had to send a message on the network, and we are not allowed to send two copies of the message. Now, assume that the job kind of dies midway, if it is restarted, another worker will again send the same set of network messages, which will cause a problem.

Hence, to ensure that there are no third parties affected, we typically prefer side effect free jobs, even though Condor can execute jobs with side effects. So, but then of course, a higher layer, a higher application layer has to take care of the problems, then the tracking module can also replicate work items. Again, for side effect free jobs, where if let us say a given work item is crucial.

For example, it is possible that the DAG of jobs is like this, that job J 1 needs to be done first and then we can start J 2, J 3, J 4 and so on. So, given the criticality of J 1, it might be a good idea to run multiple copies of J 1 and use the results of that copy which completed first. So, this will speed up the entire system, because J 1 is on the critical path.

(Refer Slide Time: 29:03)

Overview  
Design of Condor  
Detailed Operation

Main Modules  
Condor Pools  
Match Making  
Problem Solver

### Problem Solver – DAGMan

- Jobs are specified as a DAG (directed acyclic graph)
- Pre and post processing supported
- If a given job fails (because of the system or because of a bug)
  - DAGMan prints a **rescue DAG** *side-effect free*
- It is possible to have a RETRY command

NPTEL

Smruti R. Sarangi Distributed Batch Processing 21/29

So, the jobs are specified as a DAG directed acyclic graph like, you run J 1 first and then J 2, then J 3. And then of course, you might have that okay, after J 3, you run J 4, but that J 4, but for J 4 to run you have to finish J 2 and J 3 first.

So, in this way, you can specify a directed acyclic graph of jobs and Condor will respect those constraints. And furthermore, pre and post processing of jobs is supported, which means that if I have a given job, I can run a small program before the job and a small program after the job. So, what do they do? Well, the first program essentially ensures that the inputs for the job are in order.

And the second program for post processing that ensures that the outputs are in order. It also verifies pre and post, it also verifies whether the job executed correctly or not. If it has not executed correctly, then the job needs to be put back in the Condor queue such that it executes once again. Now, assume that a given job fails assume that J 1 works correctly, J 2 works correctly, but J 3 does not run. So, because J 3 does not run J 4 also does not start. So, it is possible for Condor to print rescue DAG, which in this case, would only be J 3 and J 4.

Given that the jobs J 1 and J 2 have executed, if there is some problem in J 3, the user can fix it, and just reissue J 3 and J 4. And that would complete the entire execution. So, it is possible to have a retry command where these two jobs can be executed once again. So, in this case, we do not have to run the entire DAG, we only have to run a part of it, which are the jobs J 3 and J 4. So, a distributed system with, in a condor like system would allow us to do that as long as

of course, the jobs are side effect free. So, that will allow us to run the jobs as many times as required.

(Refer Slide Time: 31:43)

Overview  
Design of Condor  
Detailed Operation

Universes  
Data Intensive Computing  
Security

### Outline

- 1 Overview
- 2 Design of Condor
  - Main Modules
  - Condor Pools
  - Match Making
  - Problem Solver
- 3 Detailed Operation
  - Universes
  - Data Intensive Computing
  - Security

Smruti R. Sarangi | Distributed Batch Processing | 22/29

### Shadow and Sandbox

- The **Shadow** is responsible for communicating the inputs of the job to the resource
  - Input files ✓
  - Network connections ✓
  - Database connections ✓
  - Executable, arguments, environment
- A resource creates a **sandbox**
  - It needs to create the appropriate environment for the job.
  - Needs to ensure that the job cannot harm the host,
  - Needs to ensure that the host cannot harm the job, *sys-calls*
  - In some cases, it needs to marshal I/O data

*Agent for int* → *Shadow* → *Job* ← *Resource* → *Sandbox*

*shadow* [file, dbms]      *Sandbox* [get time(), networks]

Smruti R. Sarangi | Distributed Batch Processing | 23/29

So, now a little bit of the details of how exactly the process works? So, we will now discuss the shadow and the sandbox. So, recall that the agent creates a shadow process. And the resource creates a sandbox. So, what is the shadow useful for? So well, when a job is specified? It might refer to a host of things, it might refer to a host of input files, to some network connections, database connections, and anything else that is a function of the environment, including environment variables.

So, to transport the entire environment, from here to there can be very expensive. It is like moving an entire container or an entire virtual machine from an agent to a resource. And this

is very expensive, it is like a full operating system move. So, this is actually not a very good idea. So, instead, what happens is a resource creates a sandbox, and the job runs within the sandbox.

So, whenever the job makes a system call, which means that it let us say it opens a file, the sandbox intercepts the system call, sends the system called to the shadow. So, let us say given file foo or txt, the job wants to read the 10th byte. Then this information is sent to the shadow. The shadow opens foo dot txt, it reads a 10th byte and sense the 10th byte back to the sandbox. So, the sandbox creates a kind of a proxy environment for the job.

It needs to ensure the job cannot harm the host. Well, this is automatically insured, because all the system calls for the job are intercepted and sent to the shadow. So, this is ensured, it needs to ensure that the host cannot harm the job. So, this is also ensured in the sense that the main way that the host actually harms is by giving it wrong arguments for system calls. But since the results come from the shadow, and the sandbox also has checks of its own. And additionally, modern sandboxes also run on secure trusted hardware, it is not really possible for the host to harm the job and the host tries something like this it will get detected.

So, the long and short of this is that the host will not be able to tamper with the job. So, then, in some cases, well it is actually in most cases, we need to marshal and unmarshal IO data. So, marshalling basically means sending data from the sandbox to the shadow. So, basically when we send data, the process of kind of making it, machine independent is marshalling. And then again reading the data and deciphering it for the shadow is unmarshalling.

So, in this case, what happens is that the data is sent for every system call, the shadow executes the system call, and it sends it back. So, of course, modern versions of the Condor, we can, we can split this a set of system calls. So, instead of for every system call, we can say that look, the shadow will run, all the system calls for the file, maybe the database connection, these will run on the shadow.

And the sandbox can run other system calls which do not really require the shadow for example, get time. So, get time we are okay. If it gets a time on the sandbox for example, or maybe send or receive a message on the network. Even that also, we are okay subject to some limitations with these can run on the sandbox.

So, of course, as I said, modern avatars of Condor allow you to write very, very flexible scripts, and also change the source code of Condor to ensure that a certain subset of system calls actually get handled by the shadow. And the remaining subset get handled by the sandbox.

(Refer Slide Time: 36:34)

Overview  
Design of Condor  
Detailed Operation

Universes  
Data Intensive Computing  
Security

## Shadow and Sandbox

- The **Shadow** is responsible for communicating the inputs of the job to the resource
  - Input files
  - Network connections
  - Database connections
  - Executable, arguments, environment
- A resource creates a **sandbox**
  - It needs to create the appropriate environment for the job.
  - Needs to ensure that the job cannot harm the host
  - Needs to ensure that the host cannot harm the job
  - In some cases, it needs to marshal I/O data

Universe: Matching sandbox and shadow pair

Smruti R. Sarangi | Distributed Batch Processing | 23/29

So, the universe, what is the universe again? Well, the universe is a matching sandbox and shadow pair. And in this case, of course, the universe has a certain set of rules. So, what I just described is the standard universe, so you can have other universes as well. And we will discuss that. In this case, of course, the idea is that the entire environment of the shadow is not sent to the sandbox. Instead, the sandbox traps system calls. So, we will see how and it sends them to the shadow the shadow executes them and sends the results back.

(Refer Slide Time: 37:14)

Overview  
Design of Condor  
Detailed Operation

Universes  
Data Intensive Computing  
Security

## Standard Universe

- Emulates a standard Unix environment
- Provides support for I/O marshalling
  - The **shadow** runs an I/O server. It takes requests from the running job, satisfies the request at the home file system, and returns the data.
  - At compile time, user code needs to be linked with Condor libraries. They wrap the I/O system calls, and convert them to RPCs.
  - It is possible to define a virtual file system using this mechanism (how???)
- Provides support for checkpointing

*fsconf*  
*Open*  
*seek*  
*read*  
*write*

*fsconf* → *shadow*

Smruti R. Sarangi | Distributed Batch Processing | 24/29

So, the standard universe which is for Unix environment, the typical system calls that are actually, where actually data transfer takes place is IO. So, the shadow run IO server, it takes requests from the running job satisfies the request to the home file system, returns the data. So, the way that we actually do it, the simplest possible way, is that at compile time, user code is linked with Condor libraries.

So, let us say a certain file command like open or F write or seek or something, read, write and seek, which are typical file access system calls. What happens is that all the library calls that call the system calls they here or something like for example, let us say F scanner, say F scan F in C allows us to read a line from a file. So, when I link a file with the Condor library, it kind of provides a wrapper on F scanner.

So, the rapper actually ensures that there is a call to the shadow, the shadow executes that and it returns the results. So, this is how we are essentially defining a kind of virtual file system with the file system is resilient on the shadow. And every single sandbox where the job runs via marshalling these IO arguments and getting the data back, we have essentially ensured that even if the cloud node does not share the same file system, our job can still run.

This is very important. Because the reason being that let us say if some remote job is running on my desktop. It should not access my files, it should access the files from where it came from. And this mechanism precisely allows us to do that. Additionally, a standard universe will provide support for check pointing which means that periodically, the entire state of the process, which is registers, the memory contents are all check pointed and they can be restored.

(Refer Slide Time: 39:39)

Overview  
Design of Condor  
Detailed Operation

Universes  
Data Intensive Computing  
Security

## Java Universe

- The sandbox creates an environment with Java virtual machines.
- It places all necessary class and archive files in the job's classpath.
- The job is linked against a Java I/O library
  - Uses a proxy I/O interface
    - Can authenticate and pass through firewalls
    - Compatible java.io.InputStream and OutputStream

agent resource  
.java

NPTEL

Srnuti R. Sarangi Distributed Batch Processing 25/29

So, then, along with the standard universe, we have a Java universe. So, why was it necessary to create a special universe for Java code? Well, because Java code typically reads a lot of other files like Java libraries. It would not have been a good idea to read all of the, read the contents of all of these files over the network using the shadow and sandbox mechanism. So, instead a Java universe kind of places all the necessary class and archive files in the class path or the job on the remote machine as well.

So, this kind of ensures that if I have the agent over here and the resource over here for any Java program running on the resource, if it needs any library file for standard Java functions, it need not come to the agent, it will find them locally in the resource itself. So, Furthermore, any kind of the so, this is clearly the biggest advantage that all of Java's baseline boilerplate files. All of Java's baseline files are available at the resource, so the job does not have to come back to the agent. That is point number one.

The other point is that we can link the same way we do in C, we can link the job with a custom Java library. So, the Java IO interface is a wrapper on Java's input and output streams. So, that does the same for regular files, sends it to the agent, and the agent sends it back agent runs the shadow, and the resource runs the sandbox. But this can be a smart interface, which can never do authentication, pass via firewalls and so on.

(Refer Slide Time: 41:38)

Overview  
Design of Condor  
Detailed Operation

Universes  
Data Intensive Computing  
Security

## Outline

- 1 Overview
- 2 Design of Condor
  - Main Modules
  - Condor Pools
  - Match Making
  - Problem Solver
- 3 Detailed Operation
  - Universes
  - Data Intensive Computing
  - Security

Smruti R. Sarangi Distributed Batch Processing 26/29

Overview  
Design of Condor  
Detailed Operation

Universes  
Data Intensive Computing  
Security

## Data Intensive Computing

- Massive amounts of data processing can be done on Condor – biological, simulation, scientific
- Create new resource manager called **Nest**
- Condor implemented a new file transfer agent called **Stork** that can synchronize large file transfers
- Using a variety of protocols – http, ftp, and Nest, **Stork** communicates with **Nest**
- To smooth out very large data transfers, Condor adds a series of Disk Routers
- A new module called **Parrot** helps Condor communicate with all kinds of unusual storage devices.

Smruti R. Sarangi Distributed Batch Processing 27/29

Now, condor has some support for data intensive computing, where we need a lot of data for biological scientific applications. So, we create a new resource manager called nest that has file transfer agents. So, a new file transfer agent was added to Condor called stork that can synchronize large file transfers. We can also use a variety of network protocols like http and ftp.

And to smooth out very large data transfers, Condor adds a series of disk base routers that optimize the communication with the hard disk. Also, condor has a new module called parrot for communicating well, I would not use the word unusual, it should not be unconventional storage devices something like flash and so on was unconventional in those days. So, that is the why this specific word unusual was used, even though it should be unconventional.



(Refer Slide Time: 42:44)

Security

- Secure Communication
  - Condor uses a secure communication library called Cedar
  - Cedar is a wrapper for SASL, Kerberos, and other authentication protocols
- Secure Execution
  - Users are given a restricted login at the resource (no chroot feature)
  - Condor can either use the Unix nobody account
  - Even better, Condor dynamically assigns a user id to a job
  - Possible to set a domain of users, such that users have same permissions in all machines in a workgroup
- Condor has a cleanup feature that kills all processes.

Finally, a word about security. So, security is clearly a big issue in a distributed system in a condor like system. Because remote job because jobs run on untrusted machines and machines run untrusted jobs. So, condor has a secure communication library called Cedar. So, Cedar is a wrapper on all secure technologies, like SSL, SASL, Kerberos and other protocols. So, it ensures that any network traffic passes in an encrypted fashion.

Furthermore, for secure execution, the question is that let us say if a remote job is running on my machine, what permissions does it have, one of the options is that at resource users are given a very restricted login and clearly the chroot the change root feature is not there, which means that they are kind of stuck to a part of the file system and there are a very limited number of things that this login can do.

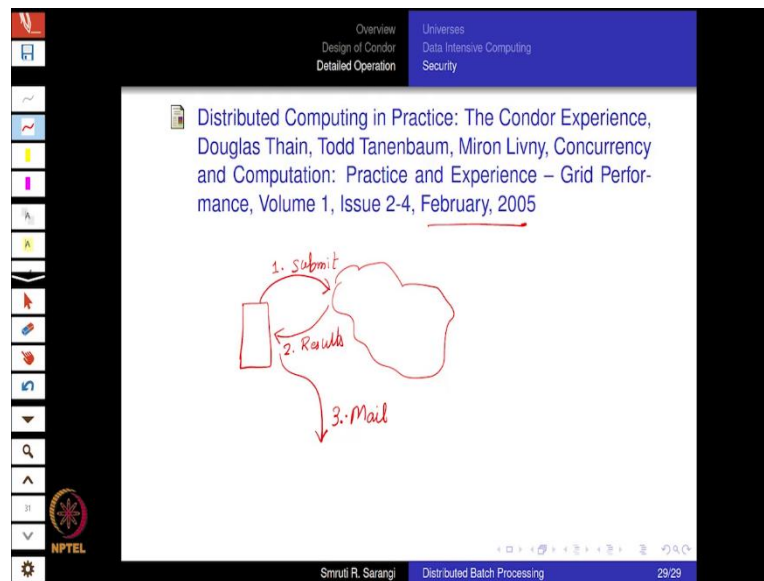
That is one option. But the other is that Unix, all Unix based systems including Linux support the nobody account. So, if you look at etc slash password, you will find the nobody account. So, nobody account is also one search extremely restrictive account that just allows us to run a compute job and access very little of the file system. So, this is actually a default, unless we have other mechanisms. Condor uses the nobody account. And which gives it very, very limited and restricted permissions on the resource.

Condor can dynamically assign a user ID to a job and create one that is also possible. Or it is possible to create a setter, create a domain of users within a group where they kind of trust each other. So, let us say that if my friend wants to run a job on my machine, and I trust my friend,

then I can create a login for my friend on my machine or my friend can use my login or we can have a standard network login. So, all have those combinations are possible.

And so, it depends on the level of trust. But as I said, nobody is a default. And creating a single domain per user like a distributed group is also possible. And finally, condor has a cleanup feature that kills all the processes. And you start from the beginning.

(Refer Slide Time: 45:28)



And so, the first description of condors, so even the condor has been around for quite some time. But the first, nicely written description of Condor the paper that this presentation is based on was the Condor experience published in February 2005. So, what I would like to encourage the viewers of this video to do, is to maybe download a condor system which is freely available, install it on their machines, and larger the cluster the better it is.

So, they will find that it is very easy to actually submit, monitor and execute a job. So, what they can do is that they can create, see if this is the Condor cloud, the machine can submit a job and particularly for those who run simulations, and let us say that they need to run 1000 simulations, these all can be sent to the Condor cloud. Condor will schedule and run the jobs.

After that, several post processing scripts can be run, which means the outputs of the jobs can be taken and graphs can be plotted and then the results can be sent back to the machine. Here also we can run a post processing script and it is possible to send an email the Job submitter to say that look, all your jobs are done and the results are ready for you to view. So, this is something which can increase productivity quite a bit in the sense that we do not have to continuously poll the system to find out how far the jobs have gone.

So, we can create a large cluster install Condor on it and automatically Condor will ensure that all the jobs correctly execute and once it is done, the user will get to know. So, this was kind of like pre 2005 technology. So, now, in the next lecture, we will discuss something which is far more recent. So, we will discuss the Microsoft Dryad link system, which kind of extends this and makes it more sophisticated.