University of Moratuwa

Department of Computer Science and Engineering

CS 4202 - Research and Development Project

Enhanced Weather Monitoring System : A Complex Event Processing and Machine Learning Based Approach

Project Group - Laridae

H. M. C. Chandrathilake (110075R)
H. T. S. Hewawitharana (110221M)
R. S. Jayawardana (110258G)
A. D. D. Viduranga (110597T)

Supervisors

Internal	Dr. H. M. N. Dilum Bandara
	Ms. Madushi Bandara
External	Dr. Srinath Perera
	Dr. Suresh Marru

Coordinated By

Dr. Malaka Walpola

THIS REPORT IS SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE AWARD OF THE DEGREE OF BACHELOR OF SCIENCE OF ENGINEERING AT UNIVERSITY OF MORATUWA, SRI LANKA.

February 05, 2016

Declaration

We, the project group Laridae (H.M.C. Chandrathilake, H.T.S. Hewawitharana, R.S. Jayawardana, A.D.D. Viduranga under the supervision of Dr. H.M.N. Dilum Bandara, Dr. Srinath Perera and Dr. Suresh Marru) hereby declare that except where specified reference is made to the work of others, the project "Complex Event Processing Based Closed Loop Weather Monitoring System" is our own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgement.

Signatures of the candidates:

1. .... H.M.C. Chandrathilake (110075R)

2. .... H.T.S. Hewawitharana (110221M)

3. .... R.S. Jayawardana (110258G)

4. .... A.D.D. Viduranga (110597T)

Supervisor:	**Project Coordinator:**
....	....
(Signature and Date)	(Signature and Date)
Dr. H.M.N. Dilum Bandara	Dr. Malaka Walpola

Abstract

Project Title:	Laridae - Enhanced Weather Monitoring System: A Complex Event Processing and Machine Learning Based Approach
Authors:	H. M. C. Chandrathilake (110075R) H. T. S. Hewawitharana (110221M) R. S. Jayawardana (110258G) A. D. D. Viduranga (110597T)
Internal Supervisors:	Dr. H. M. N. Dilum Bandara Ms. Madushi Bandara
External Supervisors:	Dr. Srinath Perera Dr. Suresh Marru

A weather forecast should be accurate and timely. Modern weather forecasting models are developed to maximize the accuracy of the forecasts by running computationally intensive algorithms with vast volumes of data. Consequently the algorithms take a long time to execute and it may adversely affect the timeliness of the forecast. One proven solution to this problem is to run the complex weather forecasting models only on the potentially hazardous events, which are pre-identified by a lightweight data filtering algorithm. Although there are a few proprietary implementations of such systems, they are built upon specific hardware-software platforms and are hard to extend or customize.

We address this problem by developing a Complex Event Processing (CEP) and Machine Learning (ML) based weather monitoring framework using open source resources which can be extended and customized according to the users' requirements. The solution consists of an open source CEP engine (WSO2-Siddhi), a ML component (Apache Spark), and an open source weather research and forecasting model (WRF). The CEP engine continuously filters out the input weather data stream to identify potentially hazardous weather events, and then generates a rough boundary enclosing all the data points within the area of interests. The filtered data points are then fed to the machine learner, where the rough boundary gets more refined. Using either K-means or Gausian Mixture Model (GMM) classifiers, the machine learner clusters areas of interests within the rough boundary given by the CEP. Finally, each cluster is processed by the complex weather algorithms of WRF model concurrently and yields the final forecast in suitable data format (e.g., GRIB). This approach significantly reduces the computational requirement, as the resource heavy weather algorithms are executed using a small subset of data that corresponds to only the areas with potentially hazardous weather events. For example, the execution time of the WRF model integrated with our solution reduced by 75% to 85%.

Acknowledgement

First and foremost we would like to express our special thanks of gratitude to our project supervisor, Dr. Dilum Bandara for his assistance, dedicated involvement and the immense support throughout the project.

We would also like to extend our sincere gratitude to Mr. Supun Nakandala and Miss. Madhushi Bandara for their help and support.

In addition, we would like to thank Dr. Srinath Perera and Dr. Surresh Marru for the valuable feedback and insights given to us regarding the project being our external supervisors.

We would like to express our warm gratitude to Dr. Malaka Walpola for coordinating the final year projects as well.

Last but not least, we would like to express our heartiest gratitude to the academic and non-academic staff and the colleagues of Department of Computer Science and Engineering, University of Moratuwa for their support.

Table of Contents

List of Figures

List of Tables

Abbreviations

API	Application Programming Interface
CASA	Collaborative Adaptive Sensing of the Atmosphere
CEP	Complex Event Processing
CLI	Command Line Interface
CSV	Comma-Separated Values
GMM	Gaussian Mixture Model
GRIB	General Regularly-distributed Information in Binary form
GUI	Graphical User Interface
HDFS	Hadoop Distributed File System
HPC	High Performance Computing
HTTP	Hypertext Transfer Protocol
JMS	Java Message Service
JVM	Java Virtual Machine
LEAD	Linked Environments for Atmospheric Discovery
ML	Machine Learning
MQTT	Message Queuing Telemetry Transport
OLAP	Online Analytical Processing
RDBMS	Relational Database Management System
RDD	Resilient Distributed Dataset
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
WRF	Weather Research & Forecasting
XSEDE	Extreme Science & Engineering Discovery

1. Introduction

1.1 Background

Throughout the history natural disasters have destroyed the lives and livelihoods, killed people, and damaged property. The damage they cause to the lives and property can be reduced by effective early warning systems. With the capability to process and analyze weather data in real time, it is possible to identify catastrophic events earlier and give timely warnings, enabling people to take necessary precautionary measures. Although many weather phenomena can be predicted based on a thorough understanding of their initial conditions, weather appears to be particularly dynamic and sensitive [1]. With the diversity of meteorological variables and sheer volume of data involved, forecasting severe weather conditions has become one of the major challenges. A large number of sensors has to be used to periodically sample atmospheric conditions that are necessary for predictions. Once collected data are transferred to the processing stations, traditional systems process them as a new batch of sensor readings. These traditional weather prediction mechanisms that use static, fixed-cycle predictions are not very effective [2].

Moreover, they usually rely on sensor data that is periodically collected without being aware of the current atmospheric conditions. These sensor readings are then processed as a batch, without any relation to results of previous analysis. It is important that the system to be a closed loop one, where the predictions will be generated while being aware of spatially and temporally correlated atmospheric events and refocusing the repeated cycles with increased sensitivity, precision and resolution. Therefore, rather than relying on periodically generated sensor readings, the closed loop will instruct sensors on what, when, and how to sense based on the detected atmospheric conditions [2]. For example, when high wind is detected on a certain region, sensors in that region will be asked to sample for wind speed, direction, and shear more frequently than compared to when the weather is calm.

It is important for the closed-loop monitoring system to be able to process these incoming streams of sensor data as they arrive than batching multiple sensor readings. This enables rapid identification of meaningful weather patterns from continuous weather data streams. Complex Event Processing (CEP) technology [3], which is used to detect patterns from large sets of incoming real-time data streams, is a better option to be used in this scenario. However, CEP alone will not be sufficient to detect weather events with complex temporal and spatial relationships. Therefore, it is important to use Machine Learning (ML) algorithms along with CEP to identify those mesoscale weather signatures. Apart from those light-weight ML algorithms, it is also required to run time consuming, complex, and proprietary weather algorithms like ARIMA [4] and WRF[12] which are more accurate. Once a significant mesoscale weather signature is detected by both CEP and light-weight ML algorithms, it is required to use those complex algorithms to confirm the weather event.

1.2 Problem Statement

Meteorological stations around the world operate continuously issuing weather predictions. For that, they use systems that can run complex weather algorithms which are capable of predicting weather events with a higher accuracy. Hadoop-based ARIMA algorithm [4] is one such example. However, these algorithms require high computational power and a longer time to produce their results. For example, the European Center for Medium Range Weather Forecast (ECMWF) system has to use supercomputers to meet the computational demands of the algorithms used to issue weather forecasts for the European region [5]. Therefore, the usual practice is to run these resource and time consuming meteorological algorithms only when there is a risk of hazardous weather phenomena. However, as these algorithms take a long time to produce the results, it may be too late to react according to the results of weather predictions. Moreover, these complex but accurate algorithms are triggered based on custom solutions/frameworks such as LEAD [6] that utilize relatively simple algorithms (e.g., thresholding) to analyze incoming weather data for potentially hazardous weather phenomena. It is not straightforward to scale these custom frameworks to handle a new meteorological variables, large number of sensors, and apply more complex detection patterns. Therefore, the specific problem that this project expects to address is:

How to use generic tools to filter out interesting weather events and identify relevant areas of interest to reduce the computational resource requirement of a closed-loop weather monitoring system?

Considering the above scenarios, we propose a system for real-time ensemble forecasts. The proposed system consists of a closed loop, CEP and ML based weather monitoring framework to continuously analyze incoming weather data streams and then to identify notable weather patterns in real time. The proposed solution consists of three phases. In the first phase, incoming data streams are continuously checked using a CEP engine to detect any preconditions that can cause hazardous weather events. While this phase can identify several simpler weather events, is can also be used to trigger the execution of relatively complex ML algorithms to detect more complex weather events. Therefore, in the second phase data are passed to appropriate ML algorithms to process further and to detect possible hazardous weather events that may exhibit complex spatial and temporal correlations. Patterns identified by the ML algorithms may also be feedback to the CEP engine to either dynamically change its detection parameters or to issue new CEP queries as and when needed. Therefore, the ability of the proposed framework to process dynamic queries is a leap forward. While more weather events can be identified at this phase, identification of several other weather events still require the execution of complex algorithms mentioned above. This is to be accomplished in the third phase based on the events trigger at first and/or second phase. This way, the complicated weather algorithms do not need to run all the time and when they are triggered to run, it will not be too late either.

1.3 Project Scope and Objectives

The project is aimed to develop a proof of concept solution to demonstrate the idea that localization of weather data processing can reduce time and resource consumption while opening the opportunity to increase the accuracy of forecasts. The concept is to be developed as a closed loop, CEP and Machine Learning based weather monitoring system. The system will be designed for the scientists to extend the scope for various weather events. For the stage, the scope of the weather events that we will specifically demonstrate is limited to thunderstorms. It will be a proof of concept to show that it is possible to predict any other desired weather event through appropriate queries. The generated forecasts can be evaluated in real-time by stakeholders such as National Weather Service forecasters, researchers via organized test bed, and can be made available to community at large. The specific objectives of the project and their scope are as follows:

- Collect streaming weather data from a set of meteorological sources and feed them into CEP engine. WSO2 Siddhi CEP engine [7] is to be used to achieve this objective because of it is a widely used CEP engine with high performance, and capable of capturing various data streams and detecting complex event patterns in real-time. Moreover, it is open source.
- Continuously run sophisticated data mining algorithms to extract potential mesoscale signatures. These algorithms are required to confirm the mesoscale signatures filtered

by the CEP engine. These ML algorithms will decide whether to execute time consuming (proprietary) weather algorithms to analyze the event further. Apache Spark [8] framework is to be used to implement these ML algorithms, as it provides a rich library of ML algorithms for big data streams which can be used to implement these algorithms.

- Trigger weather forecasting workflows in areas of detected storms. Acquire, assimilate, and interpolate observation data onto target model grids and prepare initial and lateral boundary conditions.
- Launch compute intensive ensemble models (WRF model) to further analyze the weather event. While this can be done with locally hosted WRF servers, it is also possible to use Apache Airavata [9] to execute those models remotely, as it is ideal for executing and managing time consuming computational jobs and workflows on distributed computing resources.
- Generate post processed outputs and visualizations.

1.4 Solution Overview

As the data streams from the weather sensors are fairly large and sophisticated weather prediction algorithms are very complex and consumes too much processing power and processing time, the objective of this project stands as developing a framework to manage the weather data and run tests on the data for interesting weather events. These tests can be used to run complex weather prediction algorithms later on. Also the tests can be used to configure the weather sensors for more relevant data stream on current weather events.

We named the project – ‘Laridae’, the scientific name of the ‘seagull’ as it is a common belief that appearance of seagulls near coasts indicates immediate rain.

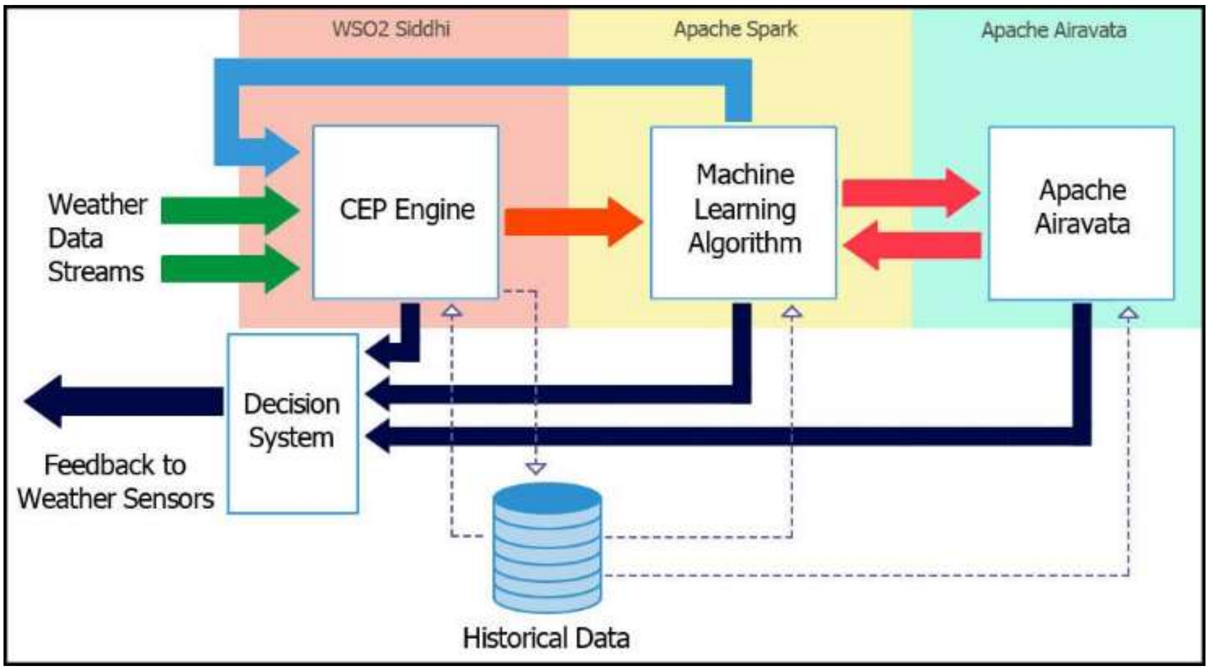


Figure 1.1 - High-level Architecture of the proposed system.

Figure1 illustrates the high-level architecture of the proposed framework. First, the data stream from the weather sensors is fed to the CEP engine. CEP engine stands as a solution to identify simple weather events (e.g., high temperature) and also a big data stream filter to the ML algorithms in the next phase In the CEP phase, the framework can identify a simple weather event and notify it to the decision system and/or send the data stream to the next phase of the framework where there are already defined ML algorithms for weather analytics. CEP filters the relevant data to be used in the ML algorithm. These filters can be dynamically modified as needed for the ML algorithms with CEP dynamic queries. In this phase, the platform can again output the results to the Decision System and/or send the data to a more sophisticated meteorological weather analytic algorithms. Historical data (archive data) part of the platform is used to get the past weather data for the decision making process in each stage of the proposed framework.

2. Literature Review

Chapter 2, the Literature Review discusses the background and technologies that are related to the project. Weather detection and particularly the thunderstorms detection are discussed, as the proposed solution is to be demonstrated using a thunderstorm-based use case. Next, in Section 2, the Weather Research & Forecasting (WRF) model for numerical weather prediction is described. CASA and LEAD, two related systems for closed-loop weather monitoring, are presented in Section 3. Related technologies that are useful for the implementation of the system are discussed next. CEP solutions which are useful in analyzing incoming weather data are presented in Section 4. Section 5 presents big-data processing frameworks. Section 6 discusses about the computing infrastructures necessary to run the complex weather detection algorithms.

2.1 Weather Events

Weather detection systems focus on hazardous and extreme weather events that could occur. There are many weather events that fall in to this category and some examples include Thunderstorms, Tornadoes, Floods, Lightning, Hail, Fog, Snowstorms, Heatwave, and Coldwave. There are many implementations of weather detection methodologies and sensor networks to support the detection of such weather events. However, detecting and tracking these weather events from real-time data streams requires very sophisticated data processing algorithms that require very high computing power. During the initial stage of the project Laridae, we will consider only the thunderstorm detection.

2.1.1 Thunderstorms

A thunderstorm is a rain shower with thunder and lightning. Although thunderstorms are most likely in the spring and summer months and during the afternoon and evening hours, they can occur year-round and at all hours. All thunderstorms can produce severe turbulence, low-level wind shear, low ceilings and visibilities, hail and lightning. Each of these hazards can be difficult to cope with; if all these conditions arrive at once, it can be disastrous.

It is estimated that 16 million thunderstorms occur in each year worldwide. In USA alone it is estimated to be around 100,000 a year and about 10% among those are considered as severe. A thunderstorm is classified as “severe”, if it contains hail (one inch or greater), winds gusting in excess of 50 knots (57.5 mph) or a tornado [10].

2.1.1.1 Formation

Thunderstorms are formed by a process called *convection*, defined as the transport of heat energy which result in upward atmospheric motion that transports whatever is in the air along with it. Because the atmosphere is heated unevenly, an imbalance can occur which thunderstorms attempt to correct. Three things are needed for convection to be a significant hazard:

- **Moisture** – Sufficient moisture must be present for clouds to form. Although convection occurs in the atmosphere without visible clouds, think thermals on a warm afternoon, moisture not only is the source of a visible cloud, but also fuels the convection to continue. As the warm air rises, it cools, and the water vapor in the air condenses into cloud droplets. The condensation releases heat, allowing the rising air to stay buoyant and continue to move upward.
- **Lift** – There are many ways for air to be lifted in the atmosphere. Convection, or buoyancy, is one method. Other meteorological methods include fronts, low pressure systems, interactions between thunderstorms, and interactions between the jet stream and the surface weather systems. Air also can be lifted by mechanical lift, such as when it is forced up and over a mountain range. Regardless of how the air is lifted, if the lift is enough to make the air warmer than the surrounding air, convection can continue.
- **Instability** – In general, as altitude increases, the air temperature cools up to the top of the troposphere. This is not always the case around fronts, mountains and in shallow layers near the ground. How fast air cools is a measure of atmospheric stability. Meteorologists refer to this vertical change in temperature as the *lapse rate*. Outside of extremes, the temperature generally decreases from between 2.7°F - 5.4°F per 1,000 feet. If the actual rising air cools slower than the lapse rate, the air remains relatively warm compared to the surroundings, and it continues to rise.

2.1.1.2 Stages of Thunderstorm Formation

As seen in Figure 1.1 meteorologists have identified three stages of thunderstorm formation:

- **Towering Cumulus Stage** – This is the stage of a thunderstorm once convection has begun and a cloud is visible. These building clouds are made entirely of liquid water. This stage is characterized by upward motion throughout the entire cloud. Aviation hazards from this stage include turbulence and icing. Even though the cloud is composed of all liquid, some of the liquid is *supercooled*, i.e., liquid water can exist at temperatures below the normal freezing point.

- **Mature Stage** – This stage is characterized by the production of precipitation. Both updrafts and downdrafts are present. Lightning is being produced. The mature thunderstorm contains water, supercooled water and ice.
- **Dissipating Stage** – During this final stage, the updraft has ceased and the storm is dominated by downdrafts. Precipitation may still occur, but will decrease with time as moisture is depleted. This dissipating thunderstorm contains mostly ice.

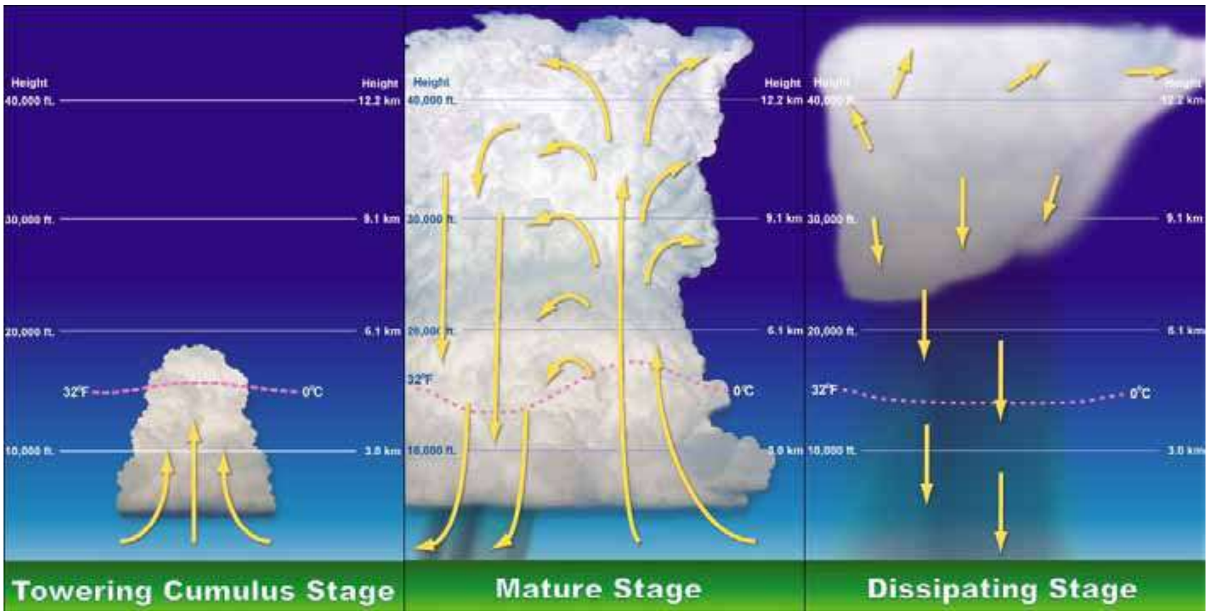


Figure 2.1 - Three stages of a Thunderstorm.

2.1.1.3 Types of Thunderstorms

Thunderstorms are classified under various types according to its severity, formation and lifetime. Following are some of the frequently occurring thunderstorm types:

- **Single-Cell Thunderstorms** – Often called "popcorn" convection, single-cell thunderstorms are small, brief, weak storms that grow and die within an hour or so. They are typically driven by heating on a summer afternoon. Single-cell storms may produce brief heavy rain and lightning.
- **Multi-Cell Storm** – A multi-cell storm is a common, garden-variety thunderstorm in which new updrafts form along the leading edge of rain-cooled air (the gust front). Individual cells usually last 30 to 60 minutes, while the system as a whole may last for many hours. Multi-cell storms may produce hail, strong winds, brief tornadoes, and/or flooding.

- **Squall Line** – A squall line is a group of storms arranged in a line, often accompanied by “squalls” of high wind and heavy rain. Squall lines tend to pass quickly and are less prone to produce tornadoes than are supercells. They can be hundreds of miles long but are typically only 15 or 30 kilometers wide.
- **Supercell** – A supercell is a long-lived (greater than 1 hour) and highly organized storm feeding off an updraft (a rising current of air) that is tilted and rotating. This rotating updraft as large as 10 miles in diameter and up to 50,000 feet tall can be present as much as 20 to 60 minutes before a tornado forms. Scientists call this rotation a mesocyclone when it is detected by Doppler radar. The tornado is a very small extension of this larger rotation. Most large and violent tornadoes come from supercells.

2.1.1.4 Mesoscale Convective System (MCS)

A Mesoscale Convective System (MCS) is a collection of thunderstorms that act as a system. An MCS can spread across an entire state and last more than 12 hours. On radar a MCS might appear as a solid line, a broken line, or a cluster of cells. This all-encompassing term can include any of the following storm types:

- **Mesoscale Convective Complex (MCC)** – A particular type of MCS, an MCC is a large, circular, long-lived cluster of showers and thunderstorms identified by satellite. It often emerges out of other storm types during the late-night and early-morning hours. MCCs can cover an entire state.
- **Mesoscale Convective Vortex (MCV)** – A low-pressure center within an MCS that pulls winds into a circling pattern, or vortex. With a core only 45 to 90 kilometers wide and 1.5 to 4.5 kilometers deep, an MCV is often overlooked in standard weather analyses. But an MCV can take on a life of its own, persisting for up to 12 hours after its parent MCS has dissipated. This orphaned MCV will sometimes then become the seed of the next thunderstorm outbreak. An MCV that moves into tropical waters, such as the Gulf of Mexico, can serve as the nucleus for a tropical storm or hurricane.

2.1.1.5 Derecho

A derecho is a widespread, long-lived wind storm that is associated with a band of rapidly moving showers or thunderstorms. Although a derecho can produce destruction similar to that of tornadoes, the damage typically is directed in one direction along a relatively straight swath. As a result, the term “straight-line wind damage” sometimes is used to describe derecho damage. By definition, if the wind damage swath extends more than 240 miles (about 400 kilometers) and includes wind gusts of at least 58 mph (93 km/h) or greater along most of its length, then the event may be classified as a derecho.

2.1.1.6 Thunderstorm Detection

Thunderstorms can be detected with variety of tools. Most of the times pictures taken from weather satellites in regular intervals from space are used to detect thunderstorms. Meteorologists watch the images of the cloud locations over time to watch for rapidly growing clouds, which would indicate a possible thunderstorm. Temperature of the clouds can also be used for forecasting thunderstorms. Clouds become cold because they are very high up in the atmosphere and the cold cloud could mean that the cloud is tall enough to be a thunderstorm. Meteorologists also use the directions of the clouds to track what areas will be affected by thunderstorm next.

Weather radars plays a very important role in forecasting because they can detect rain and severe weather events even when it is cloudy and dark. Doppler radars use electromagnetic wave and the amount of energy reflected back from the sent electromagnetic wave, can be used to predict the density of the rain and a possible hail. Doppler radars can also detect how the wind is blowing near and inside the storm which can help to understand what kind of hazards the thunderstorm might have (tornado, microburst, gust fronts, etc.) associated with it.

2.1.1.7 Thunderstorm Forecasting

Thunderstorms have a significant effect on a number of areas such as aviation, cultivation, sailing, sports and many other important domains. In addition, any individual's day-to-day work can be affected by thunderstorms. Therefore, the ability to correctly and timely forecast thunderstorms have a significant impact on the society.

Associated with this importance, there is another inevitable issue. Referred to as "Thunderstorm forecasting paradox" is the idea that over-forecasting is preferred than missing thunderstorm events. This causes an increase of false-alarm rate. But, it should be understood that weather forecasting methodologies available today have not come to perfection and weather may deviate from expected patterns at any time. Despites all these issues, there are of course a number of successful methods to predict thunderstorms. Most prominent methods include use of Computer Forecast Models, Ensemble Forecasting and Use of Satellite Data. Following is a detailed description about these technologies.

Computer Forecast Models

Numerical Weather Prediction Models are the computer programs which allows meteorologists to decide on upcoming weather events. They are algorithms that are large in complexity and consume a lot of computational power and time to run. They can be used to know how the atmosphere behaves at certain points covering large geographical areas, scaling from Earth's surface to the top of the atmosphere. These models use methods based on physics and mathematics to calculate the results and the current weather observations are used as input. The predictions are usually delivered in the form of maps, images and text.

Ensemble Forecasting

Because it is hard to gain accurate results using just one computer model, another technology of forecasting thunderstorms has been evolving where multiple computer models run simultaneously to determine whether a thunderstorm is building or not. If the results of each model agree, then it is possible to come to a concrete conclusion. If not, the causes of the misbehaviors of weather are to be found out.

Another similar way is to run the same model with different initial conditions. Running the same model several times gives multiple results giving multiple predictions and aggregating all of them, it is possible to arrive in conclusions. Interpreting the results given by these models is a key in forecasting thunderstorms and this needs a lot of practice and experience.

Satellite Data

For short-term forecasting of thunderstorms, satellite images can be used. Using them, it is possible to see where the clouds are gathered and where they are heading. Immediate patterns that are building can be understood this way. Satellites also give clues of the type of storms that can occur.

2.2 Weather Research and Forecasting Model (WRF)

Weather Research and Forecasting (WRF) Model [42] is a next-generation mesoscale weather forecasting model. It can also be considered as a data assimilation system which understands and predicts mesoscale weather and accelerates the transfer of research advances into operations. It was implemented as a multi-agency effort and can be considered as a flexible and portable code that is efficient in a massively parallel computing environment. WRF is maintained as a community model motivating to use it widely for research and teaching activities in university environments [11].

As seen in Figure 2.1 there are four components in the WRF system:

- The WRF Preprocessing System (WPS)
- WRF Variational data assimilation (WRF-Var)
- ARW solver
- Post-processing and Visualization tools

WPS is used for real data simulations. Major functionalities of WPS are, defining simulation domains, interpolating terrestrial data to the simulation domain, and degribbing and interpolating meteorological data from another model to this simulation domain. WRF-Var is optional. It is used to ingest observations into the interpolated analyses created by WPS. It can also be used to update the WRF model's initial conditions when the WRF model is run in cycling mode [12]. The WRF Software Framework (WSF) provides the infrastructure to hold several dynamics solvers, physics packages that plug into the solvers through a standard physics interface, programs for initialization and the WRF Variational data assimilation (WRF-Var) system. There are two dynamic solvers in the WSF. They are Advanced Research WRF (ARW) solver and Non-hydrostatic Mesoscale Model (NMM) solver [11].

WRF model can be used for different purposes. If both ARW and NMM dynamic solvers are used, WRF can be used for atmospheric physics and parameterization research. It also can be used for case-study research. Real time Numerical Weather Prediction (NWP) and forecast system research is another foremost usage. WRF can also be used for data assimilation research. If only ARW dynamic solver is used in the WRF, it can be used for regional climate and seasonal time-scale research. WRF is utilized for coupled chemistry applications and global simulations too [13].

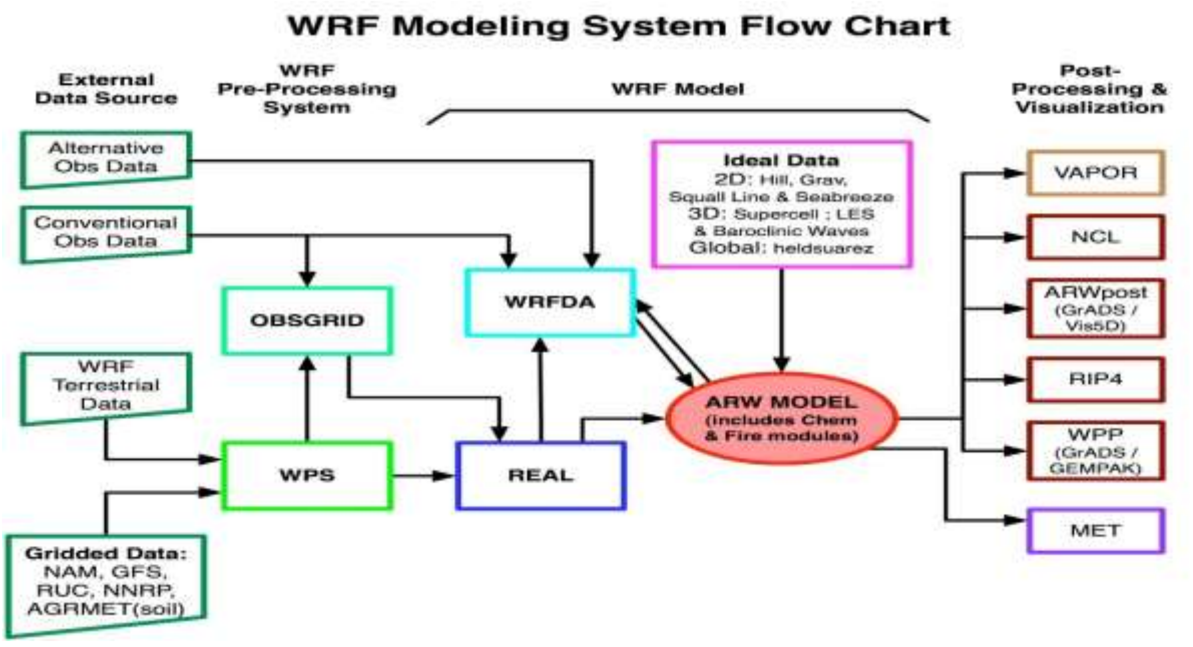

WRF Modeling System Flow Chart
External Data Source
WRF Pre-Processing System
WRF Model
Post-Processing & Visualization
Alternative Obs Data
Conventional Obs Data
WRF Terrestrial Data
Gridded Data: NAM, GFS, RUC, NNRP, AGRMET(soil)
OBSGRID
WPS
WRFDA
REAL
Ideal Data 2D: Hill, Grav, Squall Line & Seabreeze 3D: Supercell ; LES & Baroclinic Waves Global: heldsuarez
ARW MODEL (includes Chem & Fire modules)
VAPOR
NCL
ARWpost (GrADS / Vis5D)
RIP4
WPP (GrADS / GEMPAK)
MET

Figure 2.2 - WRF Modeling system flowchart [12].

WRF software framework is highly modular and is implemented with a single source code enhancing the maintainability. It is portable across a range of available computing platforms, where the model can be run on single processor, or across multiple processes with shared and distributed memory. Moreover, it supports multiple dynamics solvers and physics modules. When implementing the framework, the developers have taken care to separate the scientific codes from parallelization and other architecture specific codes. The API of WRF enables various external packages to be installed with WRF. As a result, WRF has succeeded in supporting various data formats. It is tested and guaranteed in efficient execution on range of computing platforms. It uses the Earth System Modeling Framework (ESMF) [14] timing package.

2.3 Closed-Loop Weather Detection Systems

As the weather sensing hardware and software evolve, they are generating huge volumes of sensor data streams. These data streams cannot be processed with traditional fixed-cycle weather forecasting algorithms. Closed-loop weather forecasting introduces a new way of looking at the weather data streams, and CASA and LEAD cyberinfrastructure is a good example to it.

2.3.1 Collaborative Adaptive Sensing of the Atmosphere (CASA)

Collaborative Adaptive Sensing of the Atmosphere (CASA) is weather data sensors control system which can be used to detect and predict weather events in the lowest few kilometers of the earth's atmosphere. CASA uses Distributed, Collaborative, and Adaptive Sensing (DCAS) [15] in the system which are,

- **Distributed** – Use of small radars spaced near each other to monitor the lowest few kilometers of the atmosphere despite the earth's curvature. This avoids the resolution degradation due to radar beam spreading.
- **Collaborative** – Use of multiple radars to monitor the same space in the atmosphere which achieves greater sensitivity, precision, and resolution.
- **Adaptive** – Ability of the system to dynamically reconfigure the sensors and the communication, in response to changing weather conditions. This forms a closed loop, where detected sensor data are used to drive meteorological algorithms and the findings from those algorithms are fed back to the system to change the sensing strategy to sample the weather event with high spatial and temporal resolution.

2.3.2 Linked Environments for Atmospheric Discovery (LEAD)

Linked Environments for Atmospheric Discovery (LEAD) is a middleware that facilitates adaptive utilization of distributed resources, sensors, and workflows. Lead cyberinfrastructure is based on a Service-Oriented Architecture (SOA) in which services can be dynamically reconfigured [15].

2.3.3 CASA-LEAD Cyberinfrasrtucture

CASA and LEAD are used together to develop a hardware and software framework to optimize real time, multi-scale forecasting [15]. This combination allows meteorologists to directly interact with the weather forecast streams and also control the weather sensors dynamically. As shown in Figure 3.1 CASA and LEAD cyberinfrastructure is also developed as a closed-loop between the forecast analysis and the weather sensors. The CASA system is able to dynamically reconfigure the sensors according to the outputs of LEAD to develop more focused weather data stream.

Mesoscale meteorology is the study of smaller-scale weather events such as severe storms, tornadoes, and hurricanes. System-level science in this context involves the responsiveness of the forecast models to the weather at hand, as well as conditions on the network at large and the large-scale computational resources on which forecasts rely. These responsiveness goals include:

- **Dynamic workflow adaptively** – Weather analytic workflows should be able to dynamically reconfigure in response to new events.
- **Dynamic resource allocation** – Allocate resources, including radars and weather sensors, in order to optimize data collection.
- **Continuous feature detection and data mining** – Continuously detect significant weather features and mine data to refocus detection efforts.
- **Model adaptively** – As the computational models run for a significant amount of time, they should respond to changes in weather conditions in the real time.

Figure 3.1 shows how CASA and LEAD cyberinfrastructure achieve these goals. CASA consists of an observational loop with weather data stream linking the radars to the Meteorological Command and Control (MC&C) unit. MC&C generates control messages back to the radars.

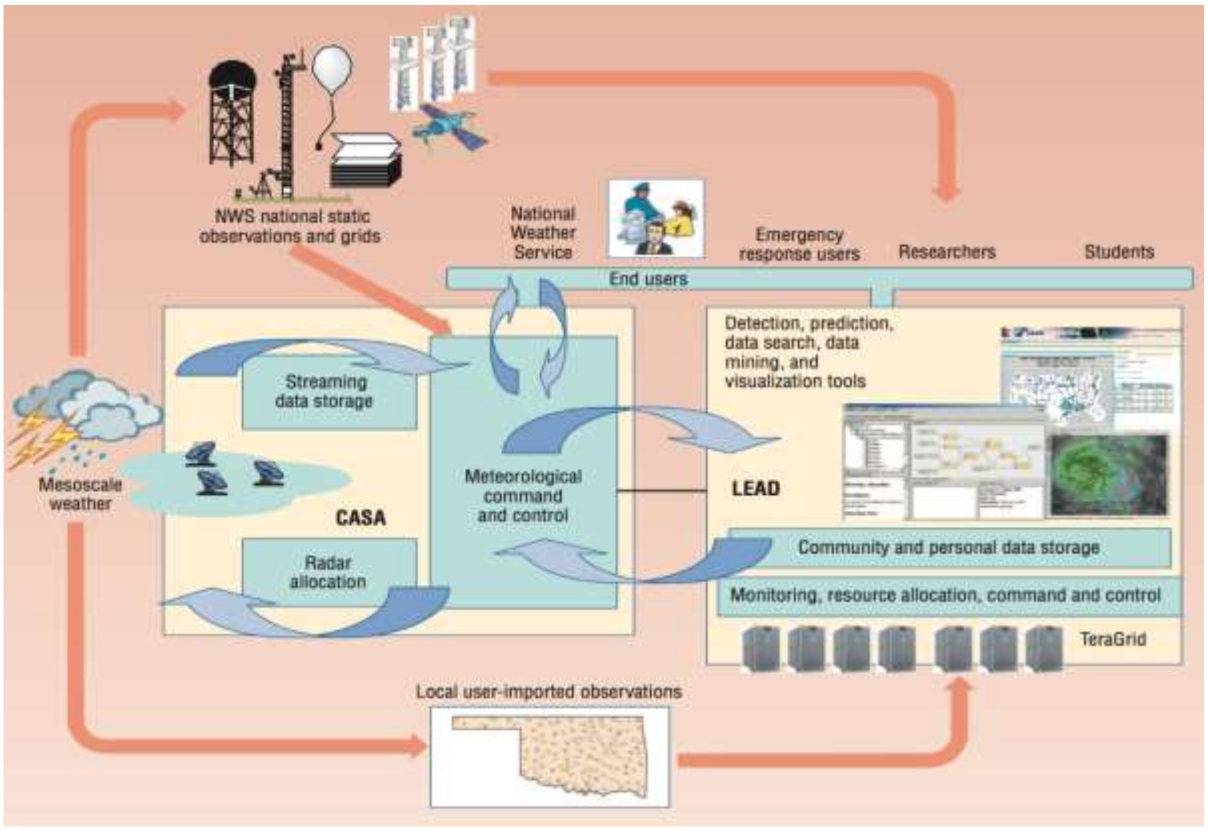

NWS national static observations and grids
National Weather Service
Emergency response users
Researchers
Students
End users
Streaming data storage
Meteorological command and control
Detection, prediction, data search, data mining, and visualization tools
Mesoscale weather
CASA
LEAD
Radar allocation
Community and personal data storage
Monitoring, resource allocation, command and control
TeraGrid
Local user-imported observations

Figure 2.3 - CASA-LEAD Cyberinfrastructure [15].

LEAD consists of a modelling loop, which executes weather analytic models in response to weather conditions. It also comprises a data storage tool to automate data staging and monitoring tool to enhance reliability and fault tolerance. LEAD closes the loop sending requests back to CASA for steering the radar location and CASA mediates potential conflicting interests from LEAD in determining the next radar position.

This helps the CASA LEAD cyberinfrastructure to achieve following responsiveness goals:

- **Dynamic workflow adaptively** – CASA uses a blackboard-based framework in which LEAD services can post messages in response to a current event. These services watch the blackboard for addressable facts within their domain of expertise. A service can select one or more facts from blackboard and propose a solution which will again be posted on the blackboard. This new fact will trigger other services.
- **Dynamic resource allocation** – CASA allocates the resources dynamically to meet user needs. The initial system accomplishes re tasking on a 30 second "heartbeat" interval based on the feedback from LEAD, the mechanically scanning radars' physical properties, and the atmospheric conditions changing rate.
- **Continuous feature detection and data mining** – CASA continuously keep detecting features in the weather data stream while LEAD can be used to dynamically mine that data to refocus detection effort
- **Model adaptively** – Model adaptively can be accomplished by launching an entirely new forecast simulation ryat finer spacing (workflow adaptation) or creating a nested grid within the simulation itself (application adaptation)

2.4 Complex Event Processing

With the advancements in computing, communication, and sensing technologies, large volumes of data are been generated every second. With the rapid increase in data volumes idea of data mining became popular. However, data mining techniques mostly rely on the processing and analysis of stored/archived data. Such techniques are not desirable in analyzing large, diverse and time sensitive data streams. Stream processing was proposed to analyze such real-time data streams about a decade ago.

Data typically originates when something of interest takes place. It may be a customer purchasing items from a super market, change in stock market values, or detection of an earthquake. In certain applications like weather and surveillance data get generated periodically and some interesting events may be buried in those streams of data. For example, sudden increase in wind speed and an intruder jumping over a fence. These origins of data are called *events* and sets of data continuously received due to the events are called *streams* [16]. Analyzing or tracking data streams (based on the events) to derive meaningful information about the events that occur is referred to as *event processing*. The sub-discipline that analyzes and tracks complex relationship among these events (based on incoming data streams) in real time is referred to as *Complex Event Processing* (CEP).

In CEP, data coming from multiple sources and of multiple formats are processed. More complicated events and patterns are then deduced. When meaningful results are identified the CEP engine (i.e., the program that performs complex event processing) reacts quickly and accordingly by generating a set of output events. These output events typically trigger the generation of an e-mail, text message, or even an alarm.

CEP combine three important aspects namely, events, Event Processing Agent (EPA), and Event Condition Action (ECA) as shown in Figure 4.1. Events are the incidents of interest as described earlier. They can be real or virtual depending on the data requirements [16]. Events can be further categorized as simple and complex events. Complex events are combinations of simple events that have some relationship among them. The EPA is responsible for filtering the events and providing them to the event processing engine. ECA defines the actions to be taken on the subject streams upon the satisfaction of defined conditions.

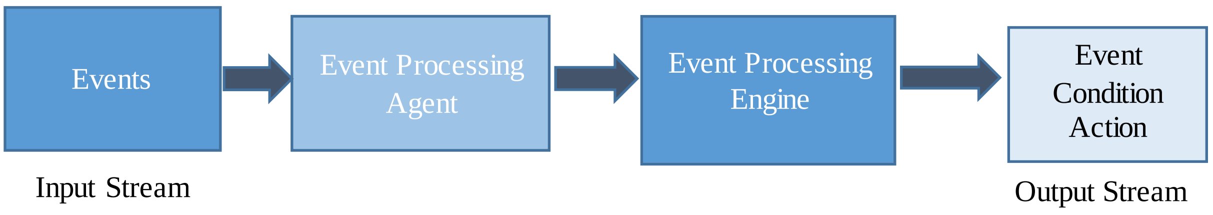


Figure 2.4 - Components of a typical CEP system.

The core processing of a CEP engine is done using the CEP execution plan. The *event processing engine* needs a specific algorithm to apply on arriving data. The developers should provide the required algorithm to the engine and it is called the *execution plan*. Typical tasks of an event processing engine include [16]:

- Event pattern detection
- Event abstraction
- Event filtering
- Event aggregation and transformation
- Modeling event hierarchies
- Detecting relationships between events
- Abstracting event-driven processes

Even though the CEP technologies are still emerging, quite a few products are already available. Some of the well-known solutions include WSO_2 Siddhi [3], Esper [17], Sybase Aleri [18], and Corel8 [19]. Next, we discuss the architectural details of some of these CEP technologies.

2.4.1 WSO2 Siddhi CEP Engine

WSO2 Siddhi is a popular open source CEP Engine. Figure 4.2 shows the data flow of the WSO_2 Siddhi CEP engine. The data streams from multiple sources arrive in different data formats. They are concatenated into a several data streams using adaptors. Next, the data streams are directed to the Siddhi query processing engine (denoted as 'Siddhi' in the diagram), which is where user defined queries are applied on data streams to identify patterns, filter, etc. The processed data are then output through output streams and they are again converted into different data types as required by the follow up users/modules. Unlike most CEP engines which use a single processing thread, Siddhi utilizes stream processing aspects like multi-threading and pipelining to gain significant performance improvement in terms of low latency and high throughput [7].

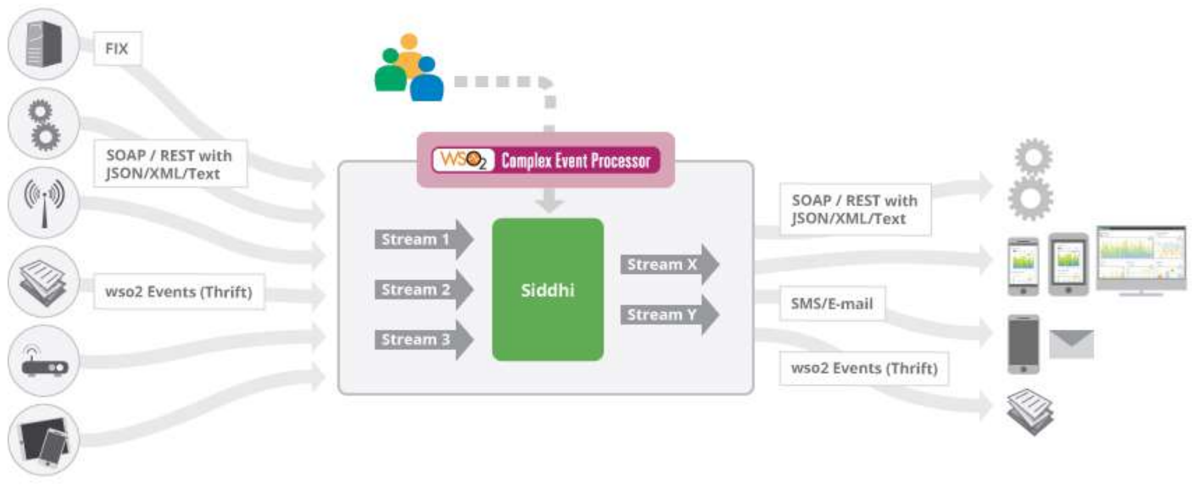


Figure 2.5 - Dataflow architecture of Siddhi CEP Engine [3].

Siddhi uses a SQL-like query language, namely Siddhi Query Language, to define input/output streams and execution plans. Siddhi queries describe how to combine existing event streams to create new event streams. When deployed in the Siddhi runtime, Siddhi queries process incoming event streams, and as specified by the queries, they generate new output event streams if they do not exist [3]. Following is a sample Siddhi query to filter high wind conditions.

```
define stream WeatherStream (latitude long, longitude long, speed
      double, direction double);
from WeatherStream [speed > 100]
select latitude, longitude
insert into outputStream;
```

2.4.2 Sybase Aleri CEP Platform

The Sybase Aleri CEP platform [18] is a complete package which includes the Aleri CEP engine as its core. To create an application on the platform, developers can use either the GUI, Splash (the query language) or Aleri Modeling language. High-level architecture of Aleri is illustrated in Figure 4.3. The complex event processor loads a data model that has been defined using one of the above mentioned options, and then waits for messages to arrive on the source streams. As each event arrives, the processing logic contained in the data model is applied to update one or more derived streams, which are the outputs. In addition to real-time streaming output, results can also be queried via the On-demand SQL query interface.

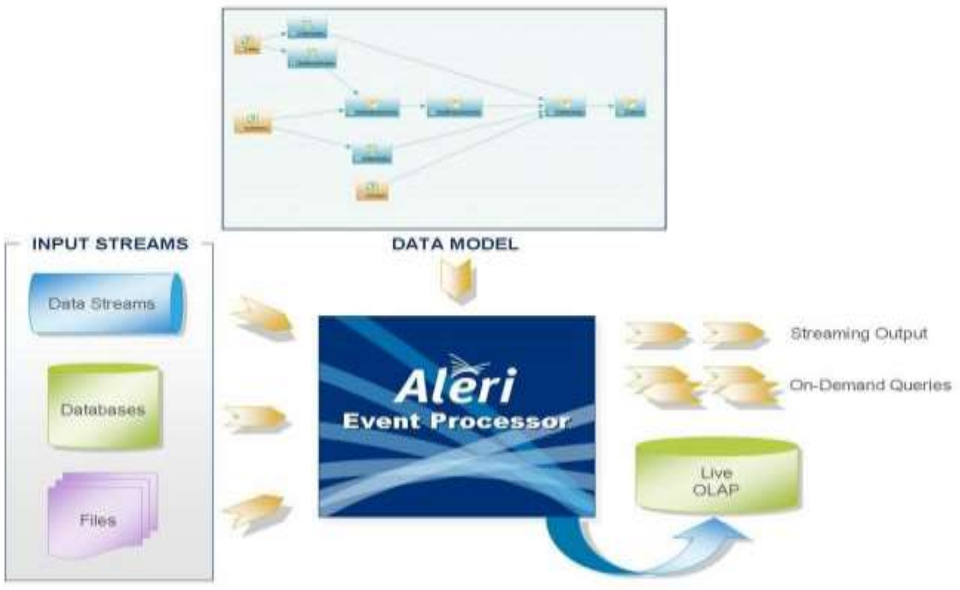


Figure 2.6 - High-level architecture of Aleri CEP Engine [18].

2.4.3 Coral8 CEP Engine

Coral8 [19] is another popular CEP engine that is designed for high volume, low-latency applications where data analysis must occur in a very short period of time. It can connect to multiple streaming data feeds and perform complex pattern matching analyses on incoming streams. It can implement execution strategies including Volume-Weighted Average Price (VWAP), Time-Weighted Average Price (TWAP), etc., Coral8 CEP is also suited to be used for other event driven solutions, including real-time dashboards, advanced visualization with Windows Presentation Foundation, Dynamic Online Analytical Processing (OLAP), event-driven business rules, real-time business process monitoring (BPM), and predictive analytics. Figure 4.4 illustrates an overview of Coral8 architecture.

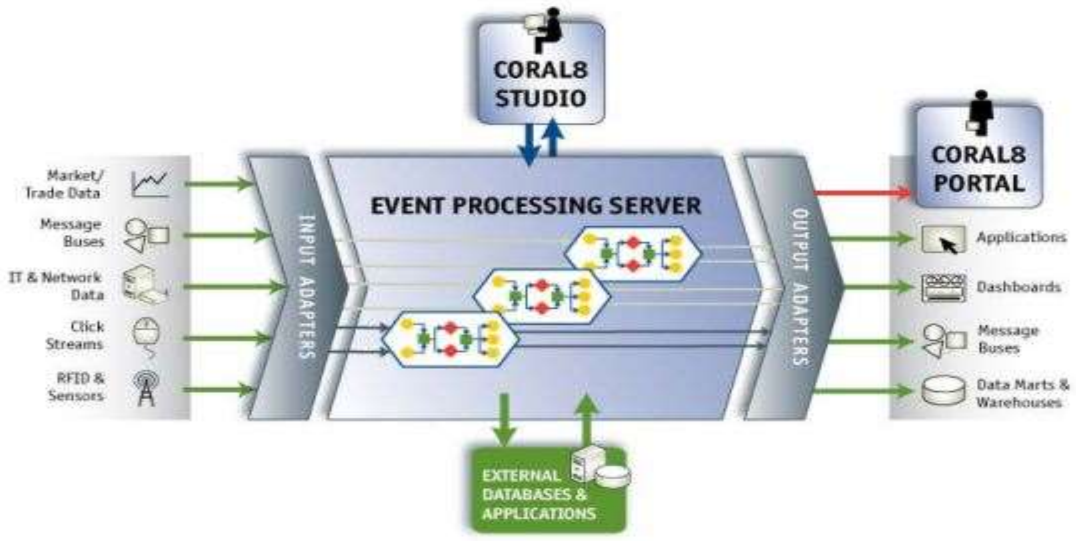


Figure 2.7 - Overview of the Architecture of Corel8 CEP [19].

The three main components of Corel8 are Server, Studio, and Portal. Coral8 Server is the high throughput, low-latency runtime server which comes with a number of packaged adapters for common high-speed data sources such as market data, messaging software, databases and more. Core8 Server facilities using external databases and applications in its operations. Coral8 Studio is an interactive graphical environment for developing, testing, and deploying Coral8 components and modules. Portal is a dashboard and visualization server that allows users to dynamically query and work with real-time CEP output.

2.4.4 Comparison of CEP Engines

When selecting a CEP engine for the project it is important to compare different CEP engines based on their applicability, technologies involved, performance, and cost. The comparison of performance is done with respect to the common set of operations shared by all the CEP engines [20], which are, Windowing, Transformation, Aggregation/Grouping, Merging, Filtering, Sorting/Ranking, Correlation and Pattern Detection. Table 4.1 and 4.2 compare the performance and other aspects of different CEP engines.

Table 1- Comparison of aspects of various CEP engines.

CEP Engine	Query Language	Platform	Product Type
Sybase Aleri	Alery SQL, similar to regular SQL	Targeted for Sybase Platform	Commercial
Corel8	CCL- an SQL base CEP language	Runs on Microsoft Windows only	Commercial
WSO2 Siddhi	Siddhi Query Language	Platform Independent	Open Source
Esper	EPL	Platform Independent	Open Source

Table 2 - Comparison of performance between Esper and Siddhi.

Tested Operation	Result
Simple filtering	Siddhi does about 20-30% better [5]
Filtering with Time Windows	Siddhi does about 20-30% better [5]
Pattern matching	Siddhi did significantly better by performing 10-15 times faster [5]
Overall performance	Overall performance of Siddhi is better than Esper [5]

Based on the comparison, Esper and WSO2 Siddhi are preferred for the project than the other two engines because they are platform independent and open source. Performance is also a very important factor for the project as the timeliness is very valuable in weather prediction. Looking at the performance comparison results of the two CEP engines, it is possible to come to the conclusion that Siddhi is the most suitable CEP engine for the project as Siddhi has a notable efficiency and performance.

2.5 Big Data Processing

2.5.1 Apache Hadoop

Apache Hadoop [23] is software framework that is used for distributed processing of large datasets across clusters of computers using simple programming models. It is an open-source software framework written in Java. Since the library is designed bearing the idea that hardware failures are common, it automatically detects and handles failures at the application layer.

There are four basic components of Apache Hadoop. *Hadoop Distributed File System* (HDFS) is the core technology for the efficient scale out storage layer. It is a distributed file-system that stores data on commodity machines. *Hadoop Common* contains libraries and utilities needed by other Hadoop modules. *Hadoop YARN* is a resource-management platform responsible for managing computing resources in clusters and using them for scheduling of users' applications. *Hadoop MapReduce* is a programming model for large-scale data processing.

2.5.2 Apache Storm

Apache Storm [1] is a free and open source distributed real-time computation system for processing large volumes of high-velocity data. Storm makes it easy to reliably process unbounded streams of data. It is simple and can be used with any programming language. Storm is extremely fast and has the ability to process over a million records per second per node on a cluster of modest size [22]. It can also be considered as a scalable system with parallel calculations that run across a cluster of machines. Storm guarantees that each unit of data (tuple) will be processed at least once or exactly once. Due to this reason, messages are only replayed when there are failures. Therefore, Storm is reliable. It is also fault tolerant and easy to operate with comparison to other frameworks [22]. A storm cluster has three sets of nodes:

- **Nimbus node** – Master node (similar to the Hadoop JobTracker) which is responsible for Uploading computations for execution, distributing code across the cluster, launching workers across the cluster, and monitoring computation and reallocates workers as needed

- **ZooKeeper nodes** – Coordinates the Storm cluster

- **Supervisor nodes** – Communicates with Nimbus through Zookeeper, starts and stops workers according to signals from Nimbus

2.5.3 Apache Spark

Apache Spark [8] is an open source, big data processing framework built around speed, ease of use, and sophisticated analytics. Spark is written in Scala programming language and runs on a Java Virtual Machine (JVM) environment. It currently supports Scala, Java, Python, Clojure and R development languages. Spark has several significant features which make it the ideal platform for big data processing and machine learning implementations:

- Less expensive shuffles in the data processing compared to Hadoop MapReduce.
- Capabilities like in-memory data storage and near real-time processing. This gives Spark better performance compared to the other big data technologies.
- Designed to be an execution engine that works both in-memory and on-disk data. Spark holds intermediate results in memory rather than writing them to disk, which is very useful especially when it is needed to work on the same data set multiple times.
- Lazy evaluation of big data queries. This optimizes steps in data processing workflows.

- Higher level API which Improves developer productivity and a consistent architect model for big data solutions.

2.5.3.1 Spark Architecture

Spark Architecture consists of three main components; Data storage, API, and Management Framework. Role of each component is as follows:

- **Data Storage** – Spark uses HDFS file system for data storage purposes. It works with any Hadoop compatible data source including HDFS, HBase, Cassandra, etc.
- **API** – The API provides the application developers to create Spark based applications using a standard API interface. Spark provides API for Scala, Java, and Python programming languages. Following are the website links for the Spark API for each of these languages.
- **Management Framework** – Spark can be deployed as a Stand-alone server or it can be on a distributed computing framework like Mesos [24] or YARN [25].

2.5.3.2 Spark Ecosystem

Other than Spark Core API, there are additional libraries that are part of the Spark ecosystem and provide additional capabilities in big data analytics and machine learning areas. These libraries include Spark Streaming [26] (for processing the real-time data), Spark SQL [27] (for running the SQL like queries), Spark MLlib [28] (a machine learning library), and Spark GraphX [29] (API for graphs and graph-parallel computation). There are also integration adapters with other products like Cassandra [30] (Spark Cassandra Connector) and R (SparkR). With Cassandra Connector, Spark can be used to access data stored in a Cassandra database and perform data analytics on that data. Figure 5.1 shows how these different libraries in Spark ecosystem are related to each other.

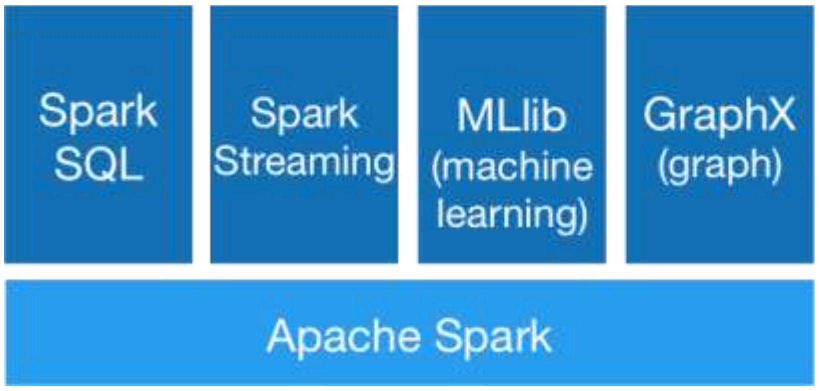


Figure 2.8 - Spark framework libraries.

2.5.3.3 Resilient Distributed Datasets

Resilient Distributed Dataset or RDD [30] is the core concept in Spark framework. Consider RDD as a table in a database. It can hold any type of data. Spark stores data in RDD on different partitions. They help with rearranging the computations and optimizing the data processing. They are also fault tolerant because an RDD know how to recreate and recompute the datasets. RDDs are immutable. An RDD can be modified with a transformation but the transformation returns you a new RDD whereas the original RDD remains the same. RDD supports two types of operations:

- **Transformation** – Transformations do not return a single value, they return a new RDD. Nothing gets evaluated when you call a Transformation function, it just takes an RDD and return a new RDD. Some of the Transformation functions are map, filter, flatMap, groupByKey, reduceByKey, aggregateByKey, pipe, and coalesce.
- **Action** – Action operation evaluates and returns a new value. When an Action function is called on a RDD object, all the data processing queries are computed at that time and the result value is returned. Some of the Action operations are reduce, collect, count, first, take, countByKey, and foreach.

2.5.3.4 Spark vs. Hadoop

As an open source framework, Apache Hadoop pioneered a fundamentally new way of storing and processing data. Instead of relying on expensive, proprietary hardware and different systems to store and process data, Hadoop enabled distributed parallel processing of huge amounts of data across inexpensive, industry-standard servers that both store and process the data, and can scale without limits. Hadoop and its MapReduce algorithm [31] have been there in the big data arena for almost a decade and have been considered as the de-facto standard for storing, processing and analyzing hundreds of terabytes, and even petabytes of data. Although Hadoop's MapReduce algorithm is a great solution for one-pass computations, it is not efficient for cases which require multi-pass computations and algorithms.

In contrast, Spark allows programmers to develop complex, multi-step data pipelines using directed acyclic graph (DAG) [32] pattern. It also supports in-memory data sharing across DAGs, so that different jobs can work with the same data. Olson at Cloudera [33] consider Apache Spark to be the leading candidate for "successor to MapReduce". Spark has several advantages over the other big data and MapReduce technologies like Hadoop [8] and Apache Storm [23]. Key advantages include the following:

- Spark gives a comprehensive, unified framework to manage big data processing requirements with a variety of datasets that are diverse in nature (e.g., text data, and graph data) as well as the source of data (batched data and real-time streaming data)
- Spark enables applications in Hadoop clusters to run up to 100 times faster in memory and 10 times faster even when running on disks.
- Spark lets the user to quickly write applications in Java, Scala, or Python. It comes with a built-in set of over 80 high-level operators which can be used interactively to query data within the shell.
- In addition to Map and Reduce operations, it supports SQL queries, streaming data, machine learning, and graph data processing. Developers can use these capabilities stand-alone or combine them to run in a single data pipeline use case.

2.5.3.5 Spark vs. Storm

There are several differences between Storm and Spark as follows:

- Storm is a stream processing framework while Spark is a batch processing framework. Both the frameworks are capable of doing micro-batching.
- Storm can work with incredibly large variety of sources while Spark work with numerous disparate sources including HDFS, Cassandra, HBase, and S3.
- Storm is mainly written with Clojure and Spouts and it is only compatible with Java, Clojure and Scala while Spark is written in Scala and API support is provided for Scala, Java and Python.
- Storm excels in the areas of near real-time analytics, natural language processing and data normalization
- Spark has an excellent model for performing iterative machine learning and interactive analytics.
- Spark performs Data-Parallel computations while Storm performs Task-Parallel computations.

Considering the above factors we can conclude that Spark is more suitable for our implementation.

2.6 Computing Infrastructures

Major resources that will be required for the functioning of the system will be discussed under this section. Light weight Machine Learning Algorithms along with CEP might not be enough to confirm the occurrence of a weather event. It is still necessary to execute time consuming, complex and proprietary weather algorithms to analyze the weather event further using compute intensive ensemble models.

Apache Airavata [34] is to be used execute weather models like *WRF Model* [35] and Airavata is ideal for executing and managing time consuming computational jobs and workflows on distributed computing resources. Apart from that, Extreme Science & Engineering Environment (XSEDE) [36] will be required which is a network of resources through which the chosen weather model will be executed.

2.6.1 Apache Airavata

Scientists often need to run complex computing programs and collect observations. First, the scientist has to collect observations and then he/she has to pass the collected data to the application and run the experiments. By automating the flow sequence (referred to as a *workflow*) without the guidance of the scientist, this task can be made easier. The solution is to use a workflow-powered science gateway to manage the experiment online. Science Gateways hide complexities in using underlying cyber infrastructure resources. They provide domain specific user interface to scientists. Apart from that, they help scientists to build communities to create experiments, share experiments and share experimental results/data.

Apache Airavata [34] is a software framework for executing and managing computational jobs and workflows on distributed computing resources including local clusters, supercomputers, national grids, and academic and commercial clouds with the goal of supporting long running applications and workflows on distributed resources. It is used to build web-based science gateways and assist to compose, manage, execute, and monitor large-scale applications (wrapped as web-based services) and workflows composed of these services [34]. It works as a gateway to bring computational resources to form a single interface for research scientists. Individual scientists can run workflows using Airavata while communities of scientists can use it through web browser interfaces. Airavata lifts the burden off the scientist by notifying the progress updates of the experiment. Figure 7.1 shows the features supported by Airavata.

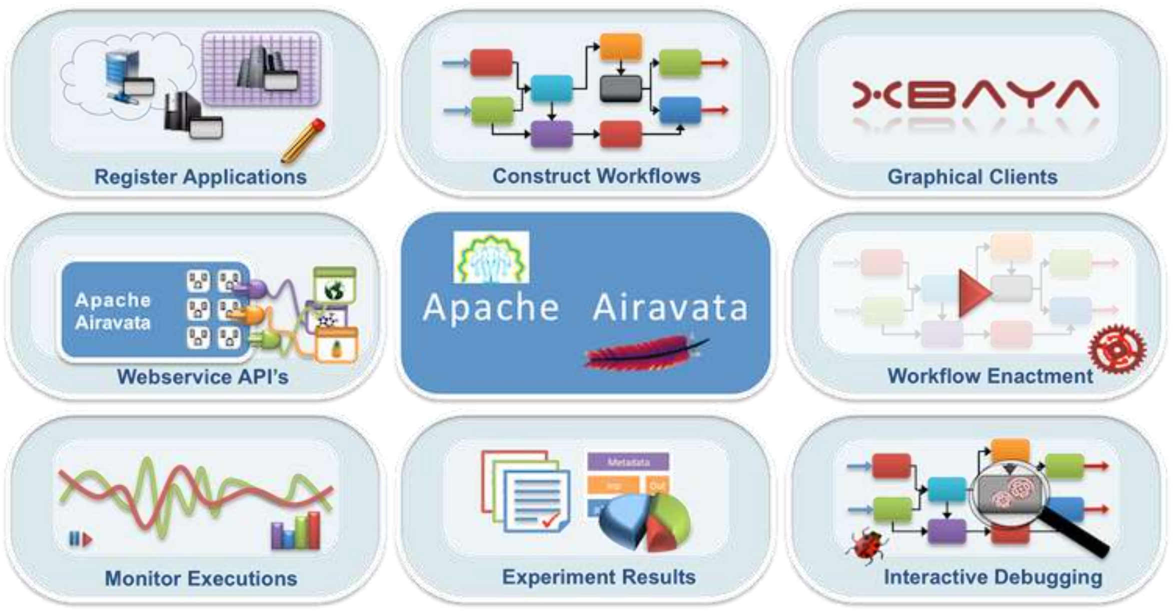


Figure 2.9 - Features of Apache Airavata [25].

As a science gateway, Apache Airavata has the following features.

- Has graphical user interfaces to construct, execute, control, manage and reuse scientific workflows.
- Has the feature of interfacing with various third party data, workflow and provenance management tools.
- Has desktop tools and browser-based web interface components for managing applications, workflows and generated data.
- Uses general-purpose industry standards and can incorporate emerging distributed and scalable infrastructures like Apache Hadoop.
- Has sophisticated server-side tools for registering and managing scientific applications on computational resources.

2.6.1.1 Apache Airavata Architecture

Airavata is built based on the Service Oriented Architecture (SOA). Apart from that, concepts like distributed messaging, and workflow composition and orchestration are also used in the implementation. The services primarily communicate using SOAP messages [34]. Figure 7.2 depicts the high-level architecture of Airavata. *XBaya* provides an easy-to-use graphical workflow composition tool for users to compose workflows from Web Services described in WSDL (Web Service Description Language). More recent releases of Airavata now use PHP Reference Gateway as a better alternative to XBaya. Registry Service provides web service interface to a database of application, service, and workflow deployment descriptions. Workflow Interpreter Service executes the workflow on one or more resources. Application

Factory Service (GFAC) manages the execution of an application in a workflow. Message Box is the WS-Notification and WS-Eventing compliant publish/subscribe messaging system for workflow events. It records the progress of the workflow execution. Airavata API is the single wrapping client that provides higher-level programing interfaces for the gateway developer [37].

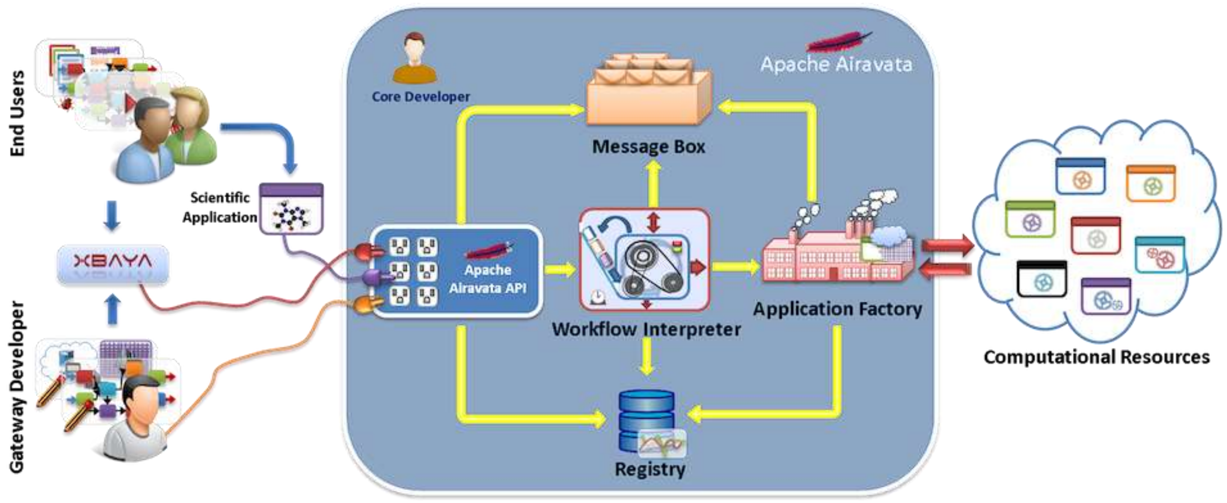


Figure 2.10 - High-level architecture of Apache Airavata [25].

2.6.2 Extreme Science and Engineering Discovery Environment (XSEDE)

Scientists around the world use advanced distributed digital resources and services each and every day to carry out their research and educational activities. They often have to use supercomputers, visualization systems, storage systems and collections of data to complete their experiments effectively. Sometimes it is hard to utilize such resources required to execute the research activities because of the high resource cost. XSEDE [36] was implemented to answer those technical barriers to access and use of digital services.

XSEDE [36] is an eco-system of advanced digital services accelerating scientific discovery. It is a single virtual system that scientists can use to interactively share computing resources, data and expertise [38]. XSEDE was implemented with the aim of deepening and extending the use of advanced digital infrastructure. It provides the access to variety of resources such as distributed memory systems, high throughput systems, visualization engines and accelerators.

XSEDE provides services such as common authentication and trust mechanisms, global namespace and file systems, remote job submissions and monitoring and file transfer services [39]. Scientists from different backgrounds can access XSEDE services, contribute resources to XSEDE and develop tools that make use of XSEDE capabilities. Users can access the

XSEDE resources by creating a user account and signing in either by using the login hub or command line (Unix/Linux).

Various software which belongs to different science categories is available on XSEDE resources supporting diverse fields of study in addition to many tools and support software including WRF (Weather Research and Forecasting) Model [34], which is essential to implement the proposed weather detection system. WRF 3.6 is the current version available in XSEDE resources.

Key benefits/advantages of XSEDE include the following:

- Availability of extensive resources that would allow a researcher to accomplish the goal. Prevailing systems like Georgia State's VELA does not support that amount of resources. Therefore larger jobs utilizing a huge memory cannot be executed in VELA.
- Availability of wide range of software on various XSEDE resources supporting diverse fields of study. Hundreds of software including visualization tools of simple 2D plots to complex 3D visualizations with high resolution rendering are supported in XSEDE.
- Availability of consistently formatted user documentation to facilitate use by the community.
- Availability of number of programming models and usage modalities on XSEDE resources. A broad range of data types and data formats are also supported.

2.6.2.1 XSEDE Architecture

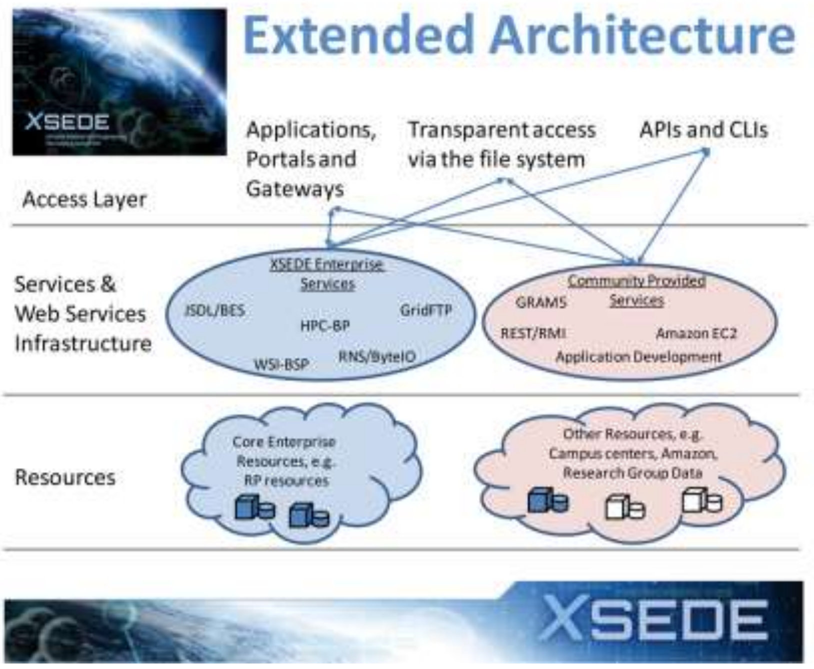

Extended Architecture
XSEDE
Applications, Portals and Gateways
Transparent access via the file system
APIs and CLIs
Access Layer
Services & Web Services Infrastructure
XSEDE Enterprise Services
JSDL/BES
HPC-BP
GridFTP
WSI-BSP
RNS/ByteIO
Community Provided Services
GRAM5
REST/RMI
Amazon EC2
Application Development
Resources
Core Enterprise Resources, e.g. RP resources
Other Resources, e.g. Campus centers, Amazon, Research Group Data
XSEDE

Figure 2.11 - XSEDE architecture [35].

Figure 7.3 depicts the XSEDE architecture. It is a combination of three different layers namely, Access Layer, Services & Web Services Infrastructure Layer and Resources Layer.

The access layer is the composition of interfaces and services including graphical user interfaces, libraries that implement application programmer interfaces (APIs), programmers that implements command line interfaces (CLIs) and file system mechanisms. The services layer can be considered as the core of the architecture. Components in this layer implements network protocol that XSEDE users can use to invoke service layer functions. The resources layer is the combination of compute servers with queuing systems, file systems, rational databases, scientific instruments, a trouble ticket database and wide area networks.

Software components of access layer connects with the software components of services layer through a well-defined set of interfaces. The services layer components interacts with the resources through resource-specific set of interfaces.

3. Design

This chapter covers all the relevant design and architectural details of the system with respect to the individual modules and their interactions and different perspectives of the stakeholders. Section 3.1 discuss the architectural constraints and design goals. Architecture of the proposed system is presented in Section 3.2. Section 3.3 presents the process view of the system and Section 3.4 presents the way in which the communication in between the system components are happening.

3.1 Architectural Constraints and Design Goals

As the problem domain is related to weather monitoring and alerting, the system is designed keeping three main design goals in mind; performance of the overall system, accuracy of the output, and extendibility of the system.

- **Performance**

  The main focus of the project is to reduce the execution time of the WRF weather forecasting module to get timely and accurate weather predictions by running WRF model with lower volume of data that are only from areas for interest. WRF module runs complex weather algorithms which consumes high computational power and execution time. In out solution, the data points are filtered out in two steps prior to feed in to the WRF module to reduce the overall processing time and excessive usage of computational resources.

- **Accuracy**

  In our solution, we cluster out the regions of interests with the filtered data points and feed them into the WRF module. Then WRF module processes the input for each region. Our system does not interact with the internal weather algorithms run in WRF module. Therefore, the accuracy of the output is totally depends on the WRF internal algorithms and is not altered by our system.

- **Extendibility**

  As a proof of concept, the system is built only to monitor the weather conditions related to the formation of thunderstorms. The system can be extended to monitor other weather events by extending the CEP and ML components. All the major modules (CEP, ML, and

WRF) are built with lose coupling and high cohesion in mind, and therefore any of the components can be replaced or modified with minimum effort, if necessary.

3.2 System Architecture

As seen in Figure 3.1 the proposed system consists of three main components; the CEP filter, machine learning clustering algorithm, and WRF module. The input weather data is given in GRIB format and is obtained from the weather data access service of National Oceanic and Atmospheric Administration, United States[41]. Though the WRF module could handle several types of NetCDF input types, we uses GRIB data format because of the ease of getting the data, compactness of the individual files, and ease of data visualization. The WRF module outputs the results in GRIB format as well.

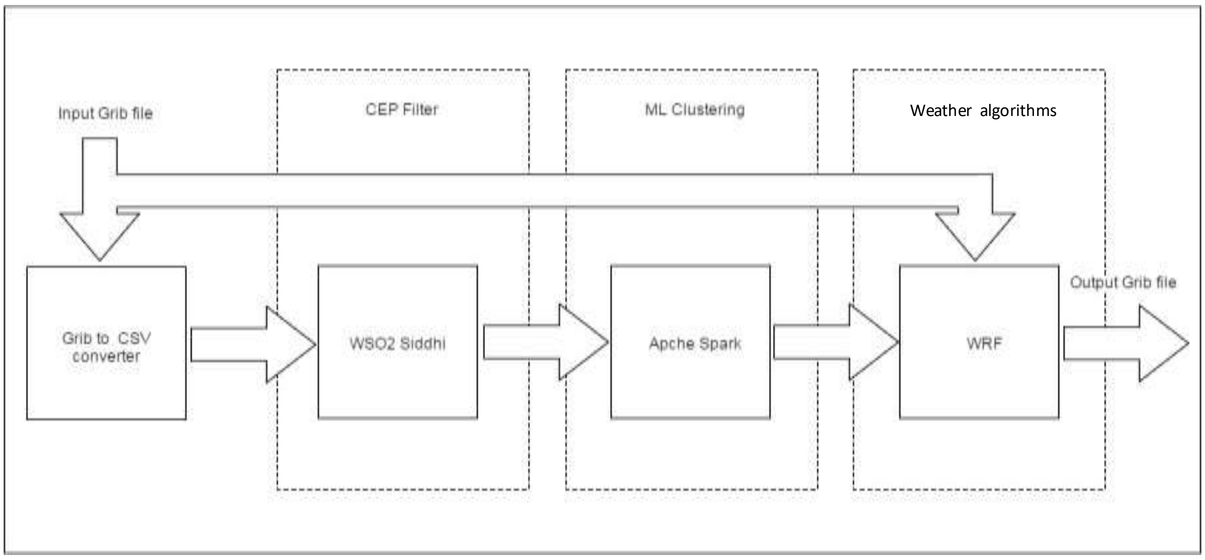


Figure 3.1 - System architecture.

The CEP engine cannot process the input GRIB files directly because the data is binary. Therefore, a GRIB to CSV conversion module is implemented to convert individual GRIB files to CSV files and feed the necessary CSV files to the CEP filter. The necessary CSV files are Best 4 layer lifted index, Storm relative helicity, and Convective inhibition. Once the above files are fed to the CEP engine through three input streams (see Section 5.4.2), the CEP engine filters out the data points from the input according to the pre-set threshold values of variables. The threshold values are used to detect thunderstorms and are explained in Section 5.4.1. At this stage, the system can create a rough boundary of the region of interest using the filtered data points.

The filtered out data points are then fed to the Machine Learner (ML) to refine the boundaries of region of interest. We use a modified version of *k*-means model and Gaussian Mixture Model

(GMM) within the ML to cluster the data points and draw regions of interest accordingly. K – means algorithm is time efficient in clustering and that lead us to choose that for the project. But, GMM algorithm gives better clusters in the context of the project. So we also used GMM for performance evaluations.

The namelist.input files of WRF module are configured with the latitude and longitude values of the identified regions of interests and is given the same input GRIB file as input so that it processes only the data bounded by the regions of interest. After the data is processed in WRF, output is given as a GRIB file and a suitable third-party tool could be used to visualize the output.

3.3 Process View

Figure 3.2 shows the system process view.

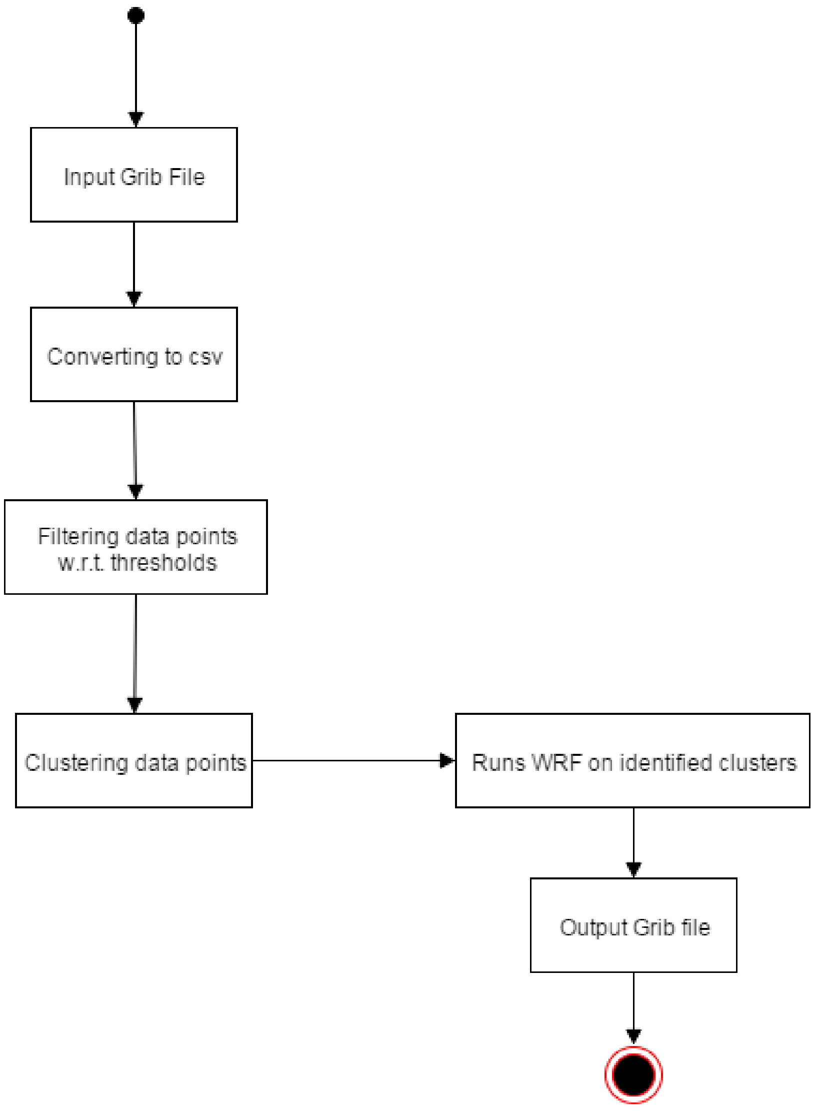

Input Grib File
Converting to csv
Filtering data points w.r.t. thresholds
Clustering data points
Runs WRF on identified clusters
Output Grib file

Figure 3.2 - System process view.

3.4 Communication between Components

Table 3.1 shows how the components of the implemented system communicate with each other. It contains the data regarding the inputs and outputs to each component.

Table 3.1 - Communication between the system components.

Module	**Input**	**Output**
CEP Filter	Best 4 layer lifted index, Storm relative helicity and Convective inhibition parameters as separate input streams in CSV format.	Latitude, longitude values of the data points filtered by the CEP rules.
Machine Learner	Latitude, longitude values of the data points filtered by CEP engine.	Latitude, longitude boundaries of each cluster (Region of interest)
WRF module	Input GRIB file	Output GRIB file

4. Implementation

This chapter discusses about the implementation of the system. Section 4.1 describes the languages and technologies used. Section 4.2 describes the implementation of the graphical user interface. Implementation details of the data feeding system is presented in Section 4.3. Implementation of the complex event processing system is described in the Section 4.4. Section 4.5 describes the implementation of the machine learning system while the WRF model integration is described in the Section 4.6.

4.1 Languages, Tools, and Technologies

This system is developed using the Java 1.8 platform. We used the Intellij IDEA as the IDE in the implementation process. We used WSO2 Siddhi 2.1.0 for the implementation of the CEP system because of its proved high performance as shown in chapter 2. For machine learning layer, we used Apache Spark (version 1.6) and its MLLib library. We used WRF version 3.7.1 ARW mode as the weather processing system/algorithm.

4.2 Graphical User Interface

Originally Laridae runs periodically capturing weather data streams and the preprocessing layers, and WRF runs sequentially without any user interaction. But for test and demonstration

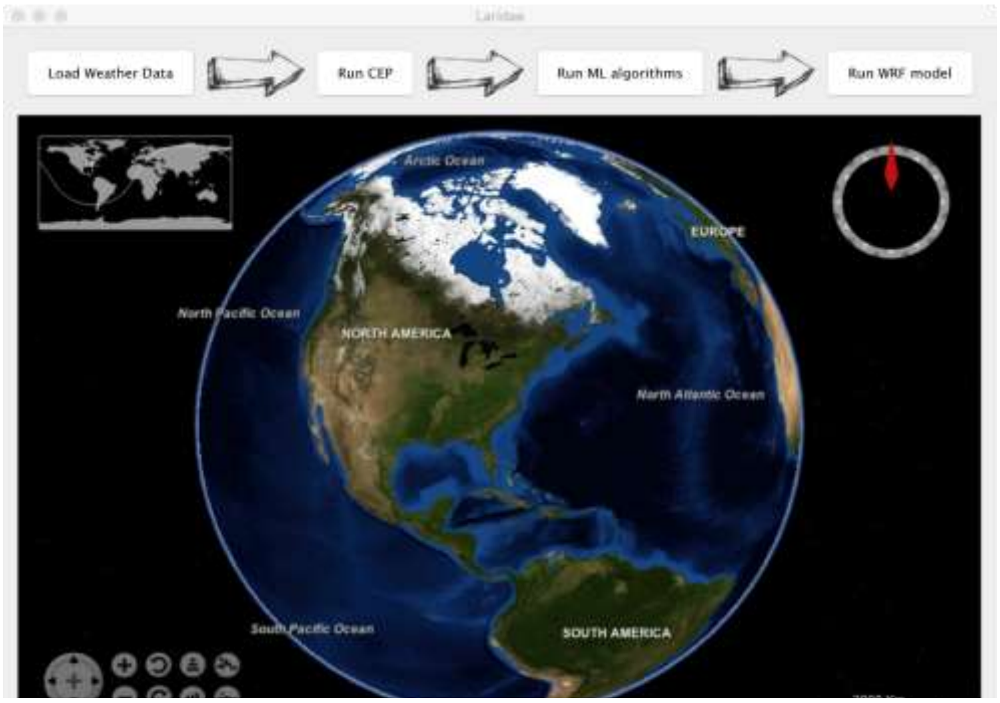


Figure 4.1 – GUI of Laridae.

purposes, a simple front end GUI is developed. Each of the process steps are broken down to handle the flow.

The Laridae process can be carried out with the sequential buttons on the top. For each Step, a visual demonstration of the processed data is shown below in the globe. WorldWind by NASA [1] is used to visualize the data on the Earth globe.

4.2.1 Load Weather Data

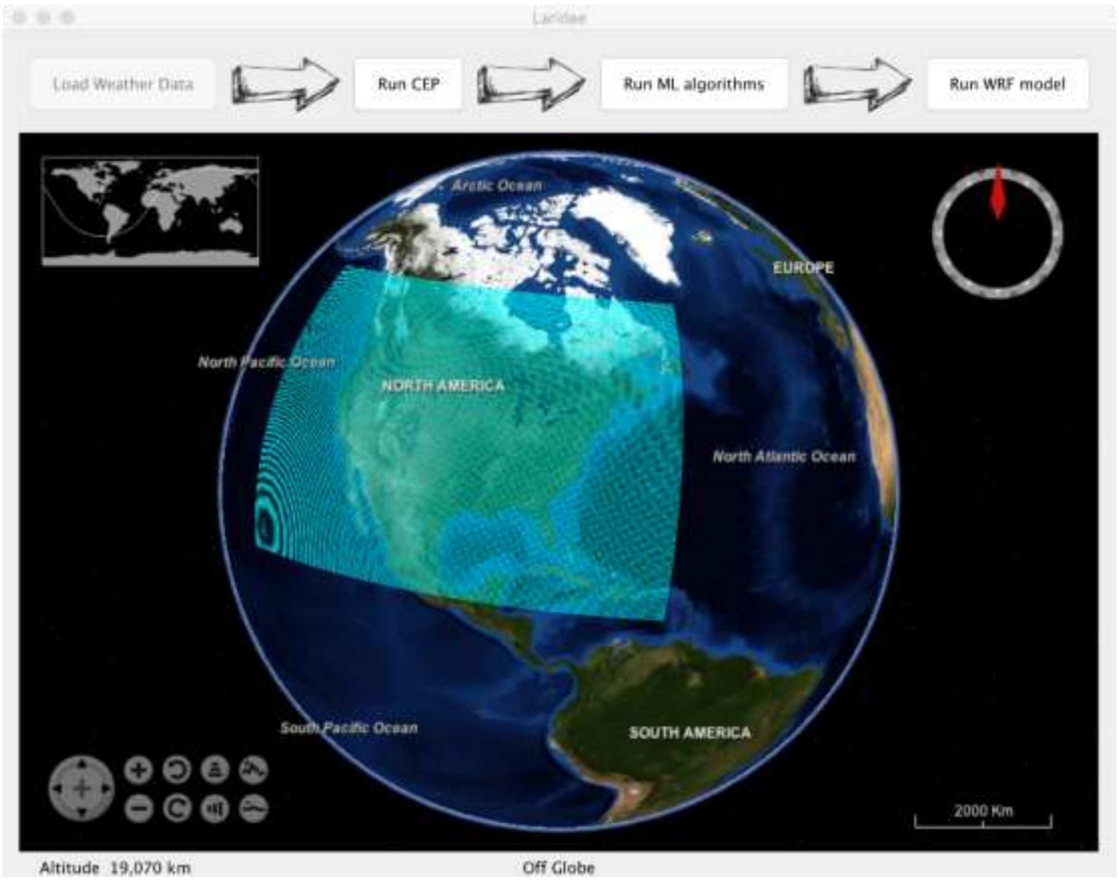


Figure 4.2 – User interface when input data is loaded.

The above Figure 4.2 shows the user interface when input data is loaded from CSV (Comma-Separated Values) format to the system. Blue colored points are the input data points to the system.

4.2.2 Run CEP

The next step of the process is to execute the siddhi cep engine and plot the filtered data points in the user interface. The below Figure 5.3 shows the user interface after completing the complex event processing to the loaded input data. The red coloured points are the filtered output from the siddhi cep engine.

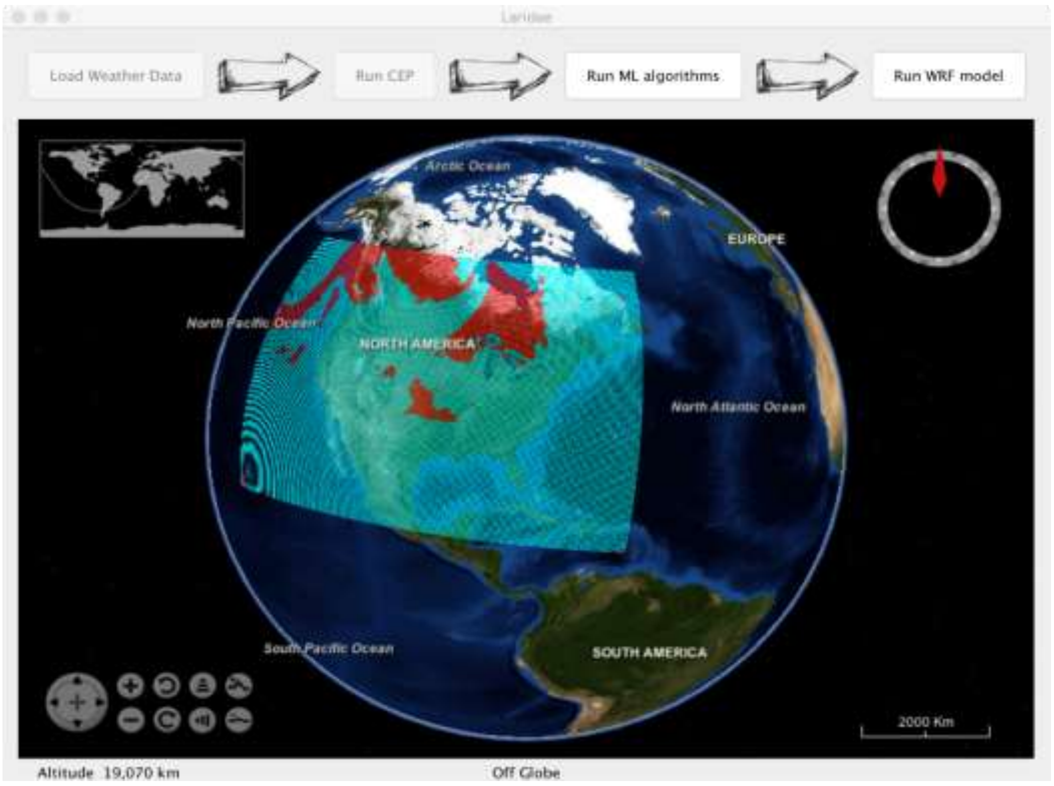


Figure 4.3 – Visualization of the CEP output.

F

4.2.1 Run ML Algorithm.

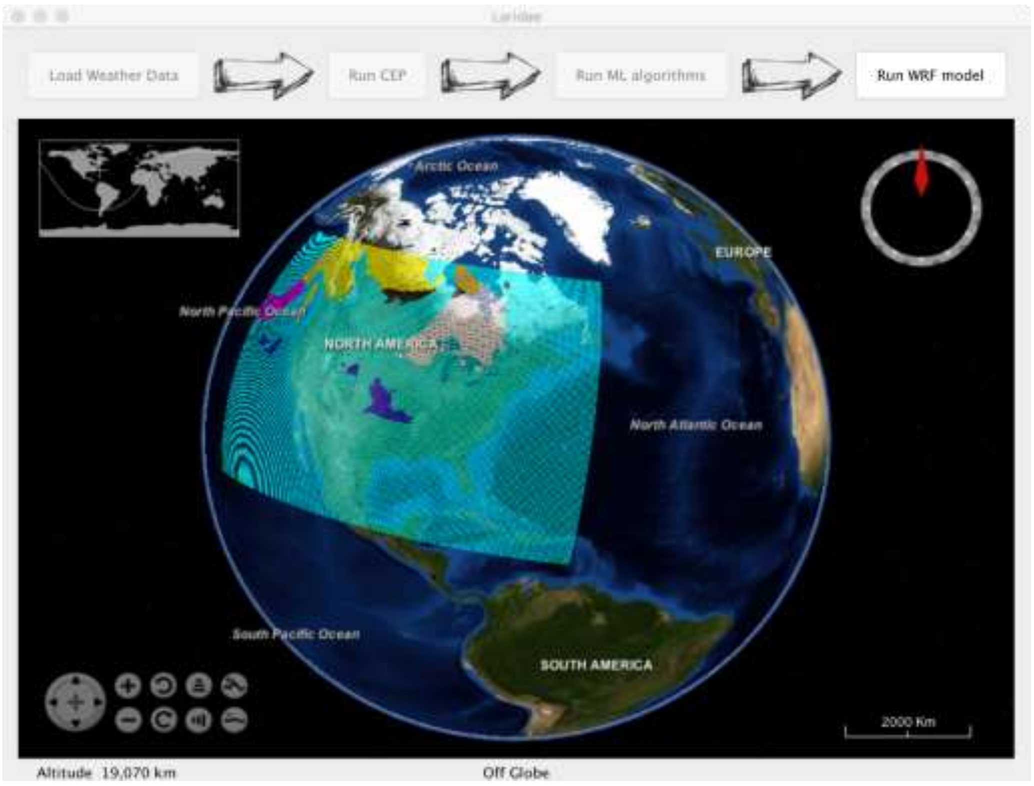


Figure 4.4 – Visualization of the ML output.

The output from the CEP engine is directed to the Machine Learning System to perform clustering. After clustering, the output clusters are displayed the user interface as in Figure 4.4 and 4.5.

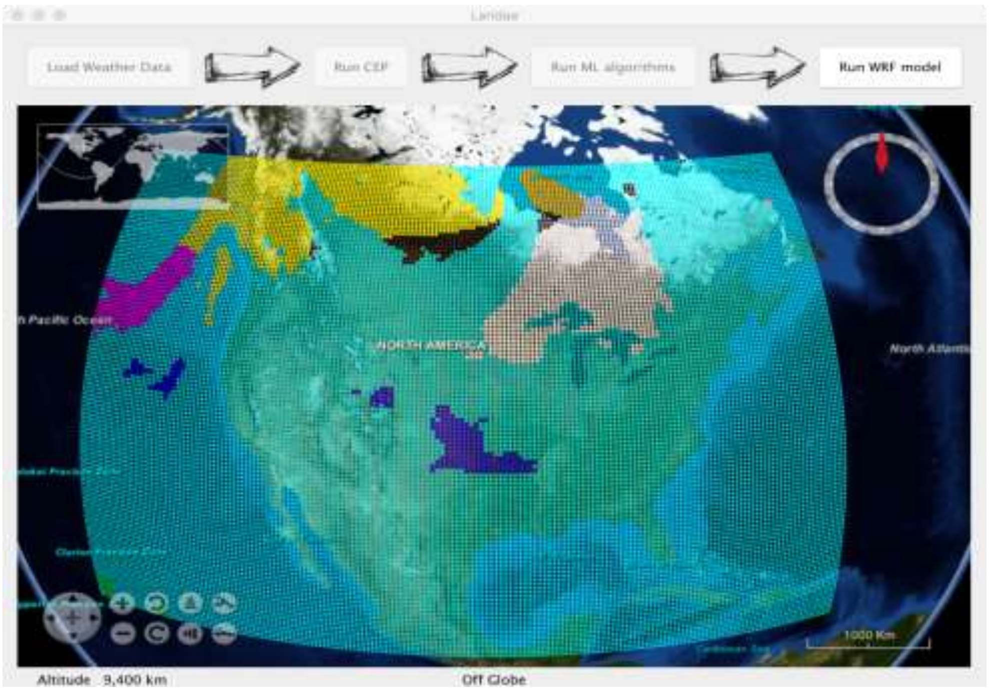


Figure 4.5 – Zoomed visualization of the ML output.

4.2.1 Run WRF.

Running WRF model ends one iteration of Laridae. Final WRF result data can then be visualized with a sophisticated tool such as Integrated Data Viewer (IDV) [2]. The below figure 4.6 shows the visualization of the output cluster data from the wrf model in IDV tool.5.

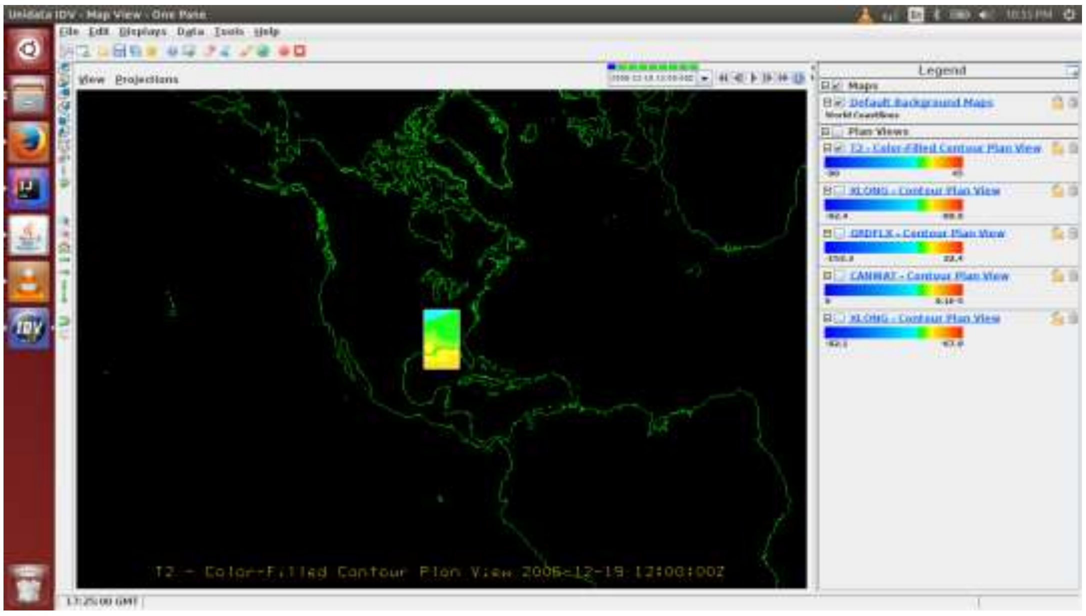

Unidata IDV - Map View - One Pane
File Edit Displays Data Tools Help
View Projections
Legend
Maps
Default Background Maps
Plan Views
T2 - Color-Filled Contour Plan View
XLONG - Contour Plan View
GRDFLX - Contour Plan View
CANWAT - Contour Plan View
XLONG - Contour Plan View
T2 - Color-Filled Contour Plan View 2006-12-19 12:00:00Z

Figure 4.6 – Visualization of WRF output.

4.3 Obtaining and Feeding Data

Weather data is obtained from the data access service of National Oceanic and Atmospheric Administration, USA. This service provides the access to a large archive of weather data in the United States, in gaps of 6 hours. The data is in GRIB (General Regularly-distributed Information in Binary form) format and is not human readable (Binary). Therefore, a visualization tool (IDV) and a conversion module (GRIB to CSV) is used to manipulate the data.

The raw GRIB data is converted into separate CSV files containing separate weather variables prior to feed to CEP engine. An open source command line tool, DEGRIB[46] is used to do this conversion. A separate module is written to automate this process.

WRF module can directly work with GRIB data and therefore the WRF module is fed with the raw GRIB data. The namelist.input files of WRF module are configured so that only the data within the specified boundaries are processed.

4.4 Complex Event Processor

Complex Event Processing System is the first part of the system. It was developed according to the system architecture described in Chapter 4. The formatted weather data in CSV format is fed into the CEP engine as the initial step. The major concern in using the CEP engine is to filter the input data by removing weather anomalies and detecting interesting data, making the processing easier in the final stages of the process. The system uses the GRIB (GRIdded Binary or General Regularly-distributed Information in Binary form) data as the input data. Different weather parameters like relative humidity, air temperature, pressure, etc. are available in the input data in GRIB format. For the data filtering few parameters were selected according to several thunderstorm indices.

4.4.1 Thunderstorm Indices

Different indices are taken into consideration to identify the probability of occurring a certain weather event like a thunderstorm. For each different thunderstorm index, different threshold values are defined in order to identify different warning levels. Using only the thunderstorm indices to predict a severe event is not successful. But for filtering the data using the defined threshold values, these parameters become useful. Thunderstorm indices can be divided into different groups like temperature and humidity based indices, wind related indices and complex parameters. For the implementation we have chosen one index per each group. Selected indices are presented next.

4.4.1.1 Lifted Index – LI

Lifted Index [43] is a temperature and humidity based index. It can be defined as the temperature difference in between the air parcel lifted adiabatically and the environment in a given pressure height in the troposphere of the atmosphere [43]. It is used to determine the stability of the lower half of the troposphere. Negative values indicate the instability of the atmosphere and it will be more buoyant the acceleration will be for rising parcels of air. When the value is more negative, air is more unstable and there is a possibility to occur convection. The thresholds are defined as follows:

Table 4.1 - Weather predictions according to the lifted index value [43].

Value of the Lifted Index	**Weather Prediction**
$LI \geq 0K$	Stable atmosphere - no thunderstorms possible
$0K > LI \geq -2K$	Thunderstorms possible
$-2K > LI \geq -6K$	Thunderstorms likely
$-6K > LI$	Severe thunderstorms likely

Accordingly, if the value obtained for the lifted index is less than or equal to zero, it can be decided that no thunderstorms are possible in the relevant area. Therefore if the lifted index value of a given location is greater than zero, we filter that location as there is a possibility of occurring a thunderstorm in that location.

4.4.1.2 Storm Relative Helicity – SRH

Storm Relative Helicity [43] belongs to the group of wind related indices and it measures the change of wind with height including its magnitude and direction in relation to the storm movement. It is calculated for the lowest 1km and 3km layers above the ground. When the value obtained for this index becomes larger, the possibility of occurring a thunderstorm event is higher. Threshold values for Storm Relative Helicity index are defined as follows:

Table 4.2 - Weather predictions according to the SRH value [43].

Value of the SRH Index	**Weather Prediction**
$SRH \leq 150\ m^2s^{-2}$	Minor rotation
$150\ m^2s^{-2} < SRH \leq 300\ m^2s^{-2}$	Thunderstorms with rotation
$300m\ m^2s^{-2} < SRH \leq 450\ m^2s^{-2}$	Supercells with rotations possible
$450\ m^2s^{-2} < SRH$	Supercells with tornadoes possible

According to the above mentioned values, if the value obtained for the SRH index is greater than 150, there is a possibility of occurring a thunderstorm. Therefore, we are using this value to filter the data. If the SRH value is less than 150, we are ignoring that data.

4.4.1.3 Convective Inhibition – CIN

Convective Inhibition [43] belongs to the category of advanced parameters. It can be defined as the energy that prevents an air parcel from rising from the surface to its level of free convection (LFC – This can be defined as the height at which lifted parcel becomes lighter than the surrounding air).

The threshold values defined for this index is as follows:

Table 4.3 - Weather Prediction according to CIN value.

Value of the CIN	**Weather Prediction**
-50 J/kg < CIN < 0	Weak
-51J/kg < CIN < -199 J/kg	Moderate
-200 J/kg < CIN	Strong

We have chosen -50 J/kg as the threshold and the input data less than the threshold are filtered out.

4.4.2 Stream Definitions

Event streams can be considered as an important part of the Siddhi CEP engine as all the processing is done based on the event streams [7]. In order to function properly, these event streams have to be defined. In implementation, we have defined three different data streams as shown in Figure 4.7.

```
define stream WeatherStream (stationId string, latitude double, longitude
double, liftedIndex double, helicity double, inhibition double)

define stream FilteredDataStream (streamId string, stationId string,
latitude double, longitude double)

define  stream  DataBoundary  (minLatitude  double,  maxLatitude  double,
minLongitude double, maxLongitude double, dataCount long)
```

Figure 4.7 – Stream definitions in Siddhi CEP Engine.

The first data stream is used to feed the input data read from CSV files to the CEP engine. We have chosen the parameters station Id, latitude of the best 4 layer lifted index, storm relative helicity and convective inhibition from the input files to be sent into the CEP engine after considering about the thunderstorm indices mentioned above. Latitude and longitude location of the weather station from which the data is recorded are also sent apart from the above parameters.

The second event stream defined above is used to feed the filtered data from the first stream. The input data in filtered using the best 4layer lifted index, storm relative helicity, and convective inhibition parameters and after filtering stream ID, station ID, latitude, and longitude data of the filtered events are inserted.

A common boundary is calculated using the filtered data from the second event stream and fed into the third event stream mentioned above. The calculated minimum and maximum latitudes and minimum and maximum longitudes are inserted as the data for the third data stream.

4.4.3 Windows

A window can be defined as a limited subset of events from an event stream [7]. If a window is defined, the events in that window can be used for calculations. There are different types of windows that could be used. For the implementation of this project we have chosen two types of windows namely length windows and time batch windows. A length window is a sliding window which keeps the last incoming *n* events while a time batch window is a time window which process data in batches [7]. Time batch windows collect incoming events arrived within the last *T* time period and gives out the output as a batch.

4.4.4 Siddhi Extensions

In instances where user defined functions have to be used apart from siddhi functionalities, Siddhi extensions are used. Siddhi supports custom codes within the queries [7]. In implementations, we have used one Siddhi extension to detect weather anomalies. When writing extensions, the Java class should be extended from the *FunctionExecutor* class. An annotation do define the namespace and function names should be added at the top of the class as follows.

```
@SiddhiExtension(namespace= "weather", function = "isNearStation")
```

Figure 4.8 – Siddhi extension definition.

Latitude and longitude locations of two points are passed to the extension as parameters. The distance between the two points are calculated using the below function and the calculated distance is compared with a threshold distance value defined. If the calculated value is less than the threshold value Boolean 'true' value is passed as the return value.

```
double theta = lonB - lonA;
double dist = Math.sin(deg2rad(latB))*Math.sin(deg2rad(latA))
      +Math.cos(deg2rad(latB))*Math.cos(deg2rad(latA))
      *Math.cos(deg2rad(theta));
dist= Math.acos(dist);
dist= rad2deg(dist);
dist= dist * 60 * 1.1515*1.609344;
if(dist < CEPEnvironment.DISTANCE_THRESHOLD) {
      isNear= true;
}
```

Figure 4.9 – Distance calculation between two points to detect anomalies.

After adding separate classes for each Siddhi extension, the extensions should be defined initially before using them in siddhi queries as follows.

```
List extensionClasses = new ArrayList();
extensionClasses.add(CEP.sidhdhiExtention.IsNearStation.class);
siddhiConfiguration siddhiConfiguration = new SiddhiConfiguration();
siddhiConfiguration.setSiddhiExtensions(extensionClasses);
siddhiManager siddhiManager = new
      SiddhiManager(siddhiConfiguration);
```

Figure 4.10 – Siddhi extension declaration.

All the siddhi extensions that are used in the project are added using an array list and it is added to the siddhi configuration. The siddhi configuration instance is passed to siddhi manager to be used in the project.

4.4.5 Execution Plans

Siddhi CEP engine uses multiple processing configurations known as execution plans. A typical execution plan consists with a set of related query expressions and those queries are defined using siddhi query language [7]. Execution plan definitions used in the implementation is discussed below.

```
"From WeatherStream [liftedIndex<"
      +CEPEnvironment.THRESHOLD_LIFTED_INDEX + "]
      #window.length(50)as A " +
"join WeatherStream[liftedIndex<"
      +CEPEnvironment.THRESHOLD_LIFTED_INDEX+"]#window.length(50)as B "+
"on weather:isNearStation
      (A.latitude,A.longitude,B.latitude,B.longitude)" +
"select 'A' as streamId, A.stationId, A.latitude, A.longitude "+
"insert into FilteredDataStream" ;
```

Figure 4.11 – Execution plan to filter data using Lifted Index.

The data is filtered using the lifted index value and it is compared with a threshold index value. If the lifted index value is less than the defined threshold value, the data is collected using a length window. The latitudes and longitudes of the filtered data are sent to a siddhi extension to detect weather it is an anomaly or not by calculating the distance between the two points. Likewise, the input data is also filtered using the storm relative helicity and the convective inhibition. The filtered values are fed into the data stream 'FilteredDataStream'.

```
"From FilteredDataStream #window.timeBatch( "
      +CEPEnvironment.TIME_GAP + " sec ) "+
"select min(latitude) as minLatitude, max(latitude) as
      maxLatitude,min(longitude) as minLongitude, max(longitude)
      as maxLongitude,count(stationId) as dataCount "+
"insert into DataBoundary for all-events" ;
```

Figure 4.12 – Boundary calculation using time batch windows.

FilteredDataStream contains the filtered data and these data are collected using a time batch window. Minimum and maximum longitudes and latitudes are calculated using the data collected inside the time batch window. The calculated values are sent to the data stream 'DataBoundary' once in a predefined time period. In this way, a common data boundary is calculated where thunderstorm events are likely to be happened.

Machine learning part of the project requires the filtered location coordinates and the common boundary to investigate data clusters. Therefore, filtered location coordinates should be filtered and passed along with the calculated boundary. This can be achieved using another siddhi extension. Siddhi supports the data types INT, LONG, BOOLEAN, FLOAT, DOUBLE and STRING [7]. The collection of location coordinates can be passed as a string. But there are instances where the coordinate string is very long and exceeds the maximum size of a string object. In java, the solution is to use either string builders or array list objects. But Siddhi CEP engine does not support them. This was solved by collecting the location coordinates using an array list inside the addCallBack method, which is called when a data is received from an event stream.

```
siddhiManager.addCallback(checkIndex,new QueryCallback() {
        public void receive(long timeStamp, Event[] inEvents,Event[]
        removeEvents){
                int k = inEvents.length;
                //adds the filtered coordinate to the array list
                if(coordString.indexOf(inEvents[k-1].getData(2)+":"+
                        inEvents[k-1].getData(3))<0){
                        coordString.append(inEvents[k-1].getData(2)+":"+
                                inEvents[k-1].getData(3)+",");
                        coordinates.add(newLocation(String.valueOf(
                                inEvents[k-1].getData(2)),String.valueOf(
                                inEvents[k-1].getData(3))));
                }
        }
});
```

Figure 4.13 - Retrieval of the output from the CEP.

It is checked whether the filtered location is previously received or not. If it is a new location which was not identified earlier, the coordinates of that point is added to an array list. After collecting the points inside the boundary, the data is sent to the machine learning system.

4.5 Machine Learning System

Machine Learning based filtering is done in the next layer of the Laridae. As a large chunks of weather sensor data used in the application in each iteration, the need of a big data processing based machine learning solution is a major requirement. To fulfill this requirement, Apache Spark Machine learning library is used in the implementation

4.5.1 Objective of ML Sub-system

CEP is used to process the weather data stream fast and give a vague boundary on a region of interest. This vague boundary boxes one or more interesting weather events across the area. The objective of the machine learning based filtering algorithm is to cluster these weather events and feed the WRF model with each individual weather events in the area.

Location coordinates of each CEP filtered data points are used to cluster them here. Using the weather sensor values along with the data point coordinates can help the machine learning based clustering algorithms to give a much reliable output. But for the scope of this project we used only binary data points to cluster and it gave relatively reliable event clusters with a good running time.

4.5.2 ML Model

Apache Spark MLLib package is still under development by the Spark Community. So it has only few ML models implemented. Currently available clustering models in Apache Spark are as follows:

- K-means
- Gaussian mixture model
- Power iteration clustering (PIC)
- Latent Dirichlet allocation (LDA)
- Bisecting k-means
- Streaming k-means

K-means and Gaussian mixture model can both be considered as reliable clustering algorithms for this application. Both of these models are implemented to be used in Laridae for clustering the CEP data points (GMM is not complete – see description below)

4.5.2.1 K-Means Model

K-means is one of most commonly used clustering algorithm that clusters the given data points to a predefined number of clusters (k). Let us have n data points $x_i, i = 1 \dots n$ that needed to be clustered in k clusters. Algorithms assigns a cluster to each data point by finding optimal position $\mu_i, i = 1 \dots k$ of the cluster centers that minimize the square of the distance from the data points to the cluster center. According to equation 4.1, K-Means algorithm minimizes the following constraint:

$$\arg min_c \sum_{i=1}^{k} \sum_{x \in c_i} d(x, \mu_i)^2 = \arg min_c \sum_{i=1}^{k} \sum_{x \in c_i} \|x, \mu_i\|_2^2 \qquad (4.1)$$

where c_i is the set of points that belong to the cluster i [40].

Apache Spark MLLib includes the above k-means implementation which can be used out of the box.

```
SparkConf sparkConf = new
      SparkConf().setAppName("JavaKMeans").setMaster("local");
JavaSparkContext sc = new JavaSparkContext(sparkConf);
JavaRDD<Vector> points = sc.parallelize(vals);

KMeans kmeans = new KMeans();
kmeans =
      kmeans.setEpsilon(epsilon).setK(k).setMaxIterations(iterations)
      .setRuns(runs).setInitializationMode(KMeans.K_MEANS_PARALLEL());

KMeansModel model = kmeans.run(points.rdd());
JavaRDD<Integer> values = model.predict(points);
```

Figure 4.14 – Implementation of K Means Algorithm.

Figure 4.14 – Implementation of K Means Algorithm.

Finding the optimal k value for the k-means algorithm is a key requirement in this process. Here we used an optimal k finding algorithm developed by D T Pham S S Dimov, and C D Nguyen [41]. The algorithm is as follows:

Find F_K s.t.

$$F_K = \begin{cases} 1\ if\ K = 1 \\ \frac{S_K}{\alpha_K S_{K-1}}\ if\ S_{K-1} \neq 0,\ \forall K > 1 \\ 1\ if\ S_{K-1} = 0,\ \forall K > 1 \end{cases} \qquad 4.2 \qquad (4.2)$$

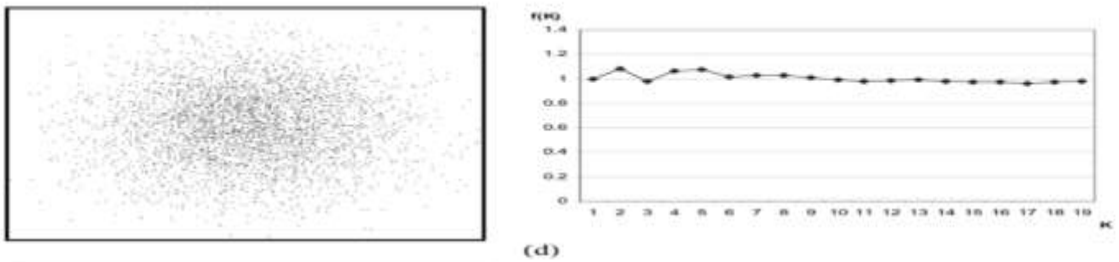
Where,

$$S_k = \sum_{j=1}^{K} I_j \longrightarrow 4.3$$

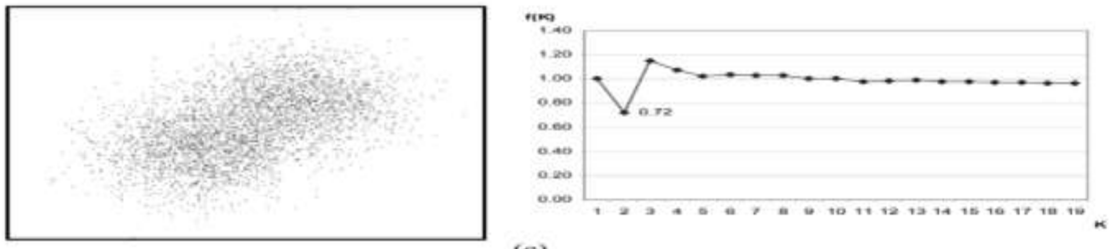
$$\alpha_K = \begin{cases} 1 - \frac{3}{4N_d}\ if\ K = 2\ and\ N_d > 1 \\ \alpha_{K-1} + \frac{1-\alpha_{K-1}}{6}\ if\ K > 2\ and\ N_d > 1 \end{cases} \longrightarrow 4.4$$

Distortion of the cluster $I_j = \sum_{t=1}^{N_j} \left[d(x_{jt}, w_j)\right]^2 \longrightarrow 4.5$

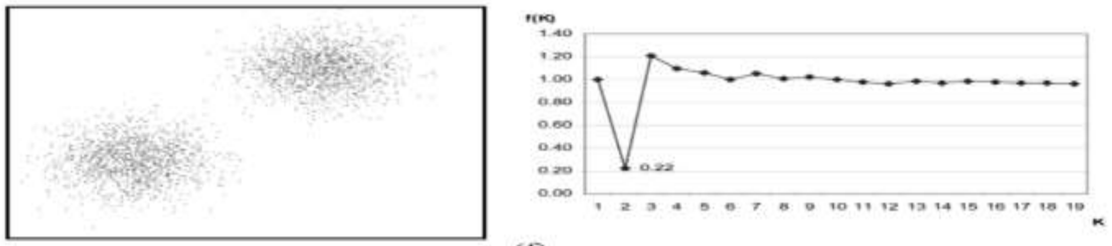
F_K results generated from the above algorithm for example data sets are illustrated in Figure 4.15. The F_K value can be successfully used to find the optimal k value for the k-means algorithm. For the implementation, the first k where $F_K < 0.85$ is used as the optimal value. See Appendix I for the implementation of the K Means algorithm.



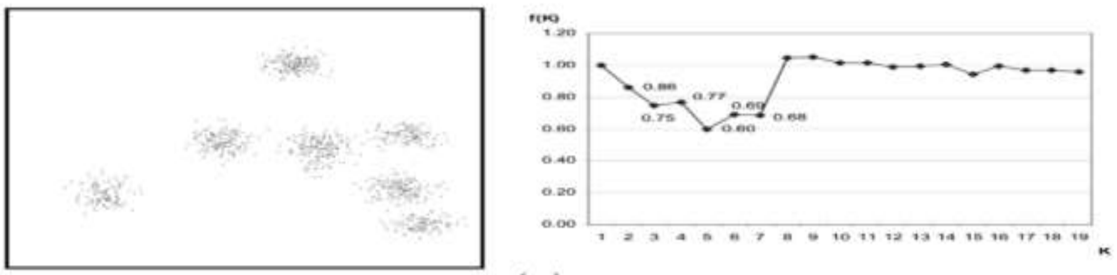
(d)



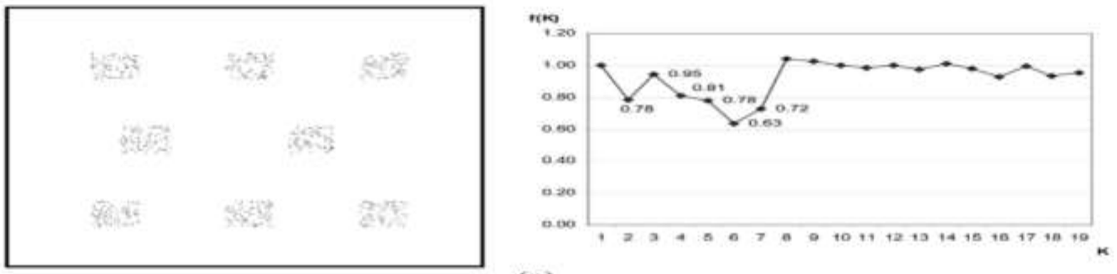
(e)



(f)



(m)



(n)

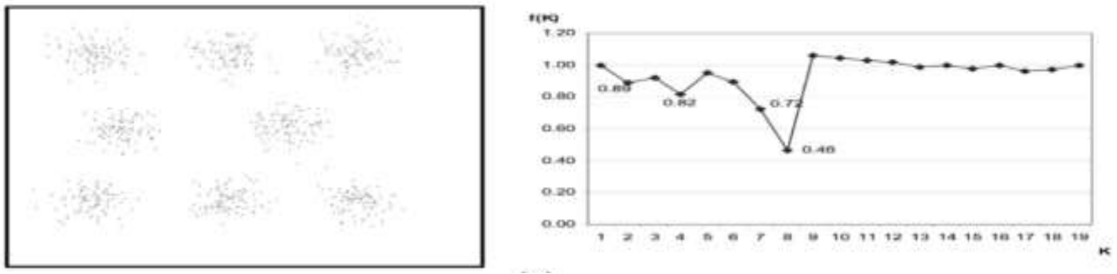


Figure 4.15 – Results generated from the Optimized K Means Algorithm.

4.5.2.1 Gaussian Mixture Model

As easy as optimizing the k-means algorithm is, it also has some problems in the weather data point filtering. One problem is that k-means algorithm assumes that all the clusters are circular. It is also problematic when the cluster sizes significantly differ. Gaussian Mixture Model gives more promising result and reduces the above problems in ML layer of Laridae.

A Gaussian Mixture Model (GMM) is a parametric probability density function represented as a weighted sum of Gaussian component densities. GMM parameters are estimated from training data using the iterative Expectation-Maximization (EM) algorithm or Maximum A Posteriori (MAP) estimation from a well-trained prior model.

Spark MLLib also implements Gausian Mixture Model. It can be used as shown in Figure 4.16

```
SparkConf sparkConf = new
SparkConf().setAppName("JavaGMM").setMaster("local");
JavaSparkContext sc = new JavaSparkContext(sparkConf);
JavaRDD<Vector> points = sc.parallelize(vals);
GaussianMixtureModel gmm = new GaussianMixture().setK(k).run(points.rdd());
JavaRDD<Integer> values = gmm.predict(points);
```

Figure 4.16 – Using Gaussian Mixture Model in implementation.

GMM also requires a predefined k value (number of clusters) which has to be determined to optimize the output clusters. This algorithm is not currently implemented in Laridae. Currently GMM algorithm relies on the following simple optimal k algorithm which determines the optimal k value where the sum of probabilities of the data points is lowest. This is not completely accurate and a more reliable algorithm has to be implemented in the future.

See Appendix I for the implementation of GMM algorithm in Apache Spark.

Akaike information criterion (AIC), Bayesian information criterion (BIC), or the Deviance information criterion (DIC) [3] can be implemented here to give GMM a better k value as future work here.

4.5.1 Objective of ML sub-system

CEP based clustering algorithm is used to process the weather data stream fast and give a vague boundary on a region of interest. This vague boundary boxes one or more interesting weather events across the area. The objective of the machine learning based filtering algorithm is to cluster these weather events and feed the WRF model with each individual weather events in the area.

Location coordinates of each CEP filtered data points are used to cluster them here. Using the weather sensor values along with the data point coordinates can help the machine learning based clustering algorithms to give a much reliable output. But for the scope of this project we used only binary data points to cluster and it gave relatively reliable event clusters with a good running time.

4.5.2 ML Model selection

Apache Spark MLLib package is still under development by the Spark Community. So it has only few ML models implemented. Currently available clustering models for Apache Spark follows,

- K-means
- Gaussian mixture
- Power iteration clustering (PIC)
- Latent Dirichlet allocation (LDA)
- Bisecting k-means
- Streaming k-means

K-means and Gaussian mixture model can both be considered as reliable clustering algorithms for this application. Both of these models are implemented to be used in Laridae for clustering the CEP data points (GMM is not complete – see description below)

4.5.2.1 K-Means Model

K-means is one of most commonly used clustering algorithm that clusters the given data points to a predefined number of clusters (k). Let us have n data points $x_i, i = 1 \dots n$ that needed to be clustered in k clusters. Algorithms assigns a cluster to each data point by finding optimal positions $\mu_i, i = 1 \dots k$ of the cluster centers that minimize the square of the distance from the data points to the cluster center. K-Means algorithm minimizes the following constraint,

$$\arg min_c \sum_{i=1}^{k} \sum_{x \in c_i} d(x, \mu_i)^2 = \arg min_c \sum_{i=1}^{k} \sum_{x \in c_i} \|x, \mu_i\|_2^2 \quad \longrightarrow \quad 4.6$$

Where c_i is the set of points that belong to the cluster i [41].

Apache Spark MLLib includes the above k-means implementation which can be used out of the box.

```
SparkConf sparkConf = new
      SparkConf().setAppName("JavaKMeans").setMaster("local");
JavaSparkContext sc = new JavaSparkContext(sparkConf);
JavaRDD<Vector> points = sc.parallelize(vals);

int k = findK(points, find_f_iterations, find_f_runs);

KMeans kmeans = new KMeans();
kmeans =
      kmeans.setEpsilon(epsilon).setK(k).setMaxIterations(iterations)
      .setRuns(runs).setInitializationMode(KMeans.K_MEANS_PARALLEL());

KMeansModel model = kmeans.run(points.rdd());
JavaRDD<Integer> values = model.predict(points);
```

Figure 4.17 – Apache Spark MLlib implementation of K Means Algorithm.

Finding the optimal k value for the k-means algorithm is a key requirement in this process. Here we used an optimal k finding algorithm developed by D T Pham, S S Dimov, and C D Nguyen [2]. The algorithm can be stated as follows.

Find F_K s.t.

$$F_K = \begin{cases} 1\ if\ K = 1 \\ \dfrac{S_K}{\alpha_K S_{K-1}}\ if\ S_{K-1} \neq 0,\ \forall K > 1 \\ 1\ if\ S_{K-1} = 0,\ \forall K > 1 \end{cases} \qquad \longrightarrow \quad 4.7$$

Where,

$$S_k = \sum_{j=1}^{K} I_j \qquad \longrightarrow \quad 4.8$$

$$\alpha_K = \begin{cases} 1 - \dfrac{3}{4N_d}\ if\ K = 2\ and\ N_d > 1 \\ \alpha_{K-1} + \dfrac{1 - \alpha_{K-1}}{6}\ if\ K > 2\ and\ N_d > 1 \end{cases} \qquad \longrightarrow \quad 4.9$$

Distortion of the cluster $I_j = \sum_{t=1}^{N_j} \left[d(x_{jt}, w_j)\right]^2 \qquad \longrightarrow \quad 4.10$

4.6 Integrating with WRF model

The process of fetching and feeding weather stream data into the system and the pre-processing work done by Complex Event Processing sub-system and the Machine Learning sub-system finally produces output in terms of refined boundaries. These information are then used to configure the WRF model and run accordingly, to analyze the data coming from inside the obtained boundaries as the final step of the dataflow. A detailed description about the implementation of this section of the system is explained in below.

4.6.1 Objective of WRF Sub-system

The ML based clustering which happens in the previous sub-system produces the boundaries of identified clusters in latitudes and longitudes. For each of these clusters, the model has to perform a localized run and the running time and the accuracy should be evaluated. To perform a localized run, two things should happen [42], they are:

- The geographical area of concern should be relatively smaller
- The resolution of the grid covering the selected area should be relatively higher

But, within the scope of this project, only the first one is considered because, it is the most relevant one in a computer science point of view, in other words, in showing that the execution time and resource consumption will be lower once a smaller area is processed. It is also a requirement to show that the actual forecast of a particular area is not missed when smaller boundaries are considered covering the same area.

For each of the clusters mentioned earlier, the execution time of the WRF model should also be gained. These are needed for the evaluation and comparison phases of the project. To get the runtime, the execution of WRF model, which consists of multiple steps, should be automated.

In addition, for each run of WRF model, the appropriate configurations must also be applied. This is mainly done through editing the files called "namelist.wps" and "namelist.input" which reside inside WPS and WRF directories respectively. Since the entire process is automated, editing the namelist files should be automated too. By this, the calculation of runtime is expected to be fair for each scenario.

4.6.2 Configuring WRF model

As mentioned earlier, the configurations required for the WRF model is given using the two namelist files. It is important to understand the content of each of them before explaining how the configuring process is automated in the system.

4.6.2.1 Content of "namelist.wps" and "namelist.input"

A sample namelist.wps file is displayed below.

```
&share
```

```
 wrf_core = 'ARW',
 max_dom = 2,
 start_date = '2008-03-24_12:00:00','2008-03-24_12:00:00',
 end_date   = '2008-03-24_18:00:00','2008-03-24_12:00:00',
 interval_seconds = 21600,
 io_form_geogrid = 2
/

&geogrid
 parent_id         =   1,   1,
 parent_grid_ratio =   1,   3,
 i_parent_start    =   1,  31,
 j_parent_start    =   1,  17,
 s_we              =   1,   1,
 e_we              =  74, 112,
 s_sn              =   1,   1,
 e_sn              =  61,  97,
 geog_data_res     = '10m','2m',
 dx = 30000,
 dy = 30000,
 map_proj = 'lambert',
 ref_lat   = 34.83,
 ref_lon   = -81.03,
 truelat1  =  30.0,
 truelat2  =  60.0,
 stand_lon = -98.,
 geog_data_path = '/mmm/users/wrfhelp/WPS_GEOG/'
/

&ungrib
 out_format = 'WPS',
prefix = 'FILE',
/

&metgrid
 fg_name = 'FILE'
 io_form_metgrid = 2,
/
```

Figure 4.18 - namelist.wps file contents

In this file, it is required to consider about several important parameters. The date and time related parameters should essentially be changed and based on the boundary information, parameters like e_we, e_sn, ref_lat, ref_lon, etc. should also be changed for each run.

After configuring the namelist.wps file, it is required to check the namelist.input file which is included inside the WRFV3 directory and change the parameters accordingly. See Appendix I for a sample namelist.input file which should be changed according to the input data.

Similar to the namelist.wps file, the most important parameters are the time and grid/location-related parameters in this file too. It is important to make sure that the values of the equivalent parameters of the two files should not conflict with each other.

Below is a description of some of the parameters found in either or both of the files [42]:

1. WRF_CORE: A character string set to either 'ARW' or 'NMM' that tells the WPS which dynamical core the input data are being prepared for. Default value is 'ARW'.
2. MAX_DOM: An integer specifying the total number of domains/nests, including the parent domain, in the simulation. Default value is 1.
3. START_YEAR: A list of MAX_DOM 4-digit integers specifying the starting UTC year of the simulation for each nest. No default value.
4. START_MONTH: A list of MAX_DOM 2-digit integers specifying the starting UTC month of the simulation for each nest. No default value.
5. START_DAY: A list of MAX_DOM 2-digit integers specifying the starting UTC day of the simulation for each nest. No default value.
6. START_HOUR: A list of MAX_DOM 2-digit integers specifying the starting UTC hour of the simulation for each nest. No default value.
7. END_YEAR: A list of MAX_DOM 4-digit integers specifying the ending UTC year of the simulation for each nest. No default value.
8. END_MONTH: A list of MAX_DOM 2-digit integers specifying the ending UTC month of the simulation for each nest. No default value.
9. END_DAY: A list of MAX_DOM 2-digit integers specifying the ending UTC day of the simulation for each nest. No default value.
10. END_HOUR: A list of MAX_DOM 2-digit integers specifying the ending UTC hour of the simulation for each nest. No default value.
11. START_DATE: A list of MAX_DOM character strings of the form 'YYYY-MM-DD_HH:mm:ss' specifying the starting UTC date of the simulation for each nest. The start_date variable is an alternate to specifying start_year, start_month, start_day, and start_hour, and if both methods are used for specifying the starting time, the start_date variable will take precedence. No default value.
12. END_DATE: A list of MAX_DOM character strings of the form 'YYYY-MM-DD_HH:mm:ss' specifying the ending UTC date of the simulation for each nest. The end_date variable is an alternate to specifying end_year, end_month, end_day, and

end_hour, and if both methods are used for specifying the ending time, the end_date variable will take precedence. No default value.

13. INTERVAL_SECONDS: The integer number of seconds between time-varying meteorological input files. No default value.
14. ACTIVE_GRID: A list of MAX_DOM logical values specifying, for each grid, whether that grid should be processed by geogrid and metgrid. Default value is .TRUE..
15. IO_FORM_GEOGRID: The WRF I/O API format that the domain files created by the geogrid program will be written in. Possible options are: 1 for binary; 2 for NetCDF; 3 for GRIB1. When option 1 is given, domain files will have a suffix of .int; when option 2 is given, domain files will have a suffix of .nc; when option 3 is given, domain files will have a suffix of .gr1. Default value is 2 (NetCDF).
16. OPT_OUTPUT_FROM_GEOGRID_PATH: A character string giving the path, either relative or absolute, to the location where output files from geogrid should be written to and read from. Default value is './'.
17. DEBUG_LEVEL: An integer value indicating the extent to which different types of messages should be sent to standard output. When debug_level is set to 0, only generally useful messages and warning messages will be written to standard output. When debug_level is greater than 100, informational messages that provide further runtime details are also written to standard output. Debugging messages and messages specifically intended for log files are never written to standard output, but are always written to the log files. Default value is 0.

5.6.2.1.2 geogrid Section

First write a sentence or 2 before listing these

1. PARENT_ID: A list of MAX_DOM integers specifying, for each nest, the domain number of the nest's parent; for the coarsest domain, this variable should be set to 1. Default value is 1

2. PARENT_GRID_RATIO: A list of MAX_DOM integers specifying, for each nest, the nesting ratio relative to the domain's parent. No default value.

3. I_PARENT_START: A list of MAX_DOM integers specifying, for each nest, the x-coordinate of the lower-left corner of the nest in the parent unstaggered grid. For the coarsest domain, a value of 1 should be specified. No default value.

4. J_PARENT_START: A list of MAX_DOM integers specifying, for each nest, the y-coordinate of the lower-left corner of the nest in the parent unstaggered grid. For the coarsest domain, a value of 1 should be specified. No default value.

5. S_WE: A list of MAX_DOM integers which should all be set to 1. Default value is 1.

6. E_WE: A list of MAX_DOM integers specifying, for each nest, the nest's full west-east dimension. For nested domains, e_we must be one greater than an integer multiple of the nest's parent_grid_ratio (i.e., *e_we* = *n* * *parent_grid_ratio* + 1 for some positive integer *n*). No default value.

7. S_SN: A list of MAX_DOM integers which should all be set to 1. Default value is 1.

8. E_SN: A list of MAX_DOM integers specifying, for each nest, the nest's full south-north dimension. For nested domains, e_sn must be one greater than an integer multiple of the nest's parent_grid_ratio (i.e., *e_sn* = *n* * *parent_grid_ratio* + 1 for some positive integer *n*). No default value.

9. GEOG_DATA_RES : A list of MAX_DOM character strings specifying, for each nest, a corresponding resolution or list of resolutions separated by + symbols of source data to be used when interpolating static terrestrial data to the nest's grid. For each nest, this string should contain a resolution matching a string preceding a colon in a rel_path or abs_path specification (see the description of GEOGRID.TBL options) in the GEOGRID.TBL file for each field. If a resolution in the string does not match any such string in a rel_path or abs_path specification for a field in GEOGRID.TBL, a default resolution of data for that field, if one is specified, will be used. If multiple resolutions match, the first resolution to match a string in a rel_path or abs_path specification in the GEOGRID.TBL file will be used. Default value is 'default'.

10. DX: A real value specifying the grid distance in the x-direction where the map scale factor is 1. For ARW, the grid distance is in meters for the 'polar', 'lambert', and 'mercator' projection, and in degrees longitude for the 'lat-lon' projection; for NMM, the grid distance is in degrees longitude. Grid distances for nests are determined recursively based on values specified for parent_grid_ratio and parent_id. No default value.

11. DY: A real value specifying the nominal grid distance in the y-direction where the map scale factor is 1. For ARW, the grid distance is in meters for the 'polar', 'lambert', and 'mercator' projection, and in degrees latitude for the 'lat-lon' projection; for NMM, the grid distance is in

degrees latitude. Grid distances for nests are determined recursively based on values specified for parent_grid_ratio and parent_id. No default value.

12. MAP_PROJ: A character string specifying the projection of the simulation domain. For ARW, accepted projections are 'lambert', 'polar', 'mercator', and 'lat-lon'; for NMM, a projection of 'rotated_ll' must be specified. Default value is 'lambert'.

13. REF_LAT: A real value specifying the latitude part of a (latitude, longitude) location whose (i,j) location in the simulation domain is known. For ARW, ref_lat gives the latitude of the center-point of the coarse domain by default (i.e., when ref_x and ref_y are not specified). For NMM, ref_lat always gives the latitude to which the origin is rotated. No default value.

14. REF_LON: A real value specifying the longitude part of a (latitude, longitude) location whose (i, j) location in the simulation domain is known. For ARW, ref_lon gives the longitude of the center-point of the coarse domain by default (i.e., when ref_x and ref_y are not specified). For NMM, ref_lon always gives the longitude to which the origin is rotated. For both ARW and NMM, west longitudes are negative, and the value of ref_lon should be in the range [-180, 180]. No default value.

15. REF_X: A real value specifying the i part of an (i, j) location whose (latitude, longitude) location in the simulation domain is known. The (i, j) location is always given with respect to the mass-staggered grid, whose dimensions are one less than the dimensions of the unstaggered grid. Default value is (((*E_WE* - 1.)+1.)/2.) = (*E_WE*/2.).

16. REF_Y: A real value specifying the j part of an (i, j) location whose (latitude, longitude) location in the simulation domain is known. The (i, j) location is always given with respect to the mass-staggered grid, whose dimensions are one less than the dimensions of the unstaggered grid. Default value is (((E_SN-1.)+1.)/2.) = (E_SN/2.).

17. TRUELAT1: A real value specifying, for ARW, the first true latitude for the Lambert conformal projection, or the only true latitude for the Mercator and polar stereographic projections. For NMM, truelat1 is ignored. No default value.

18. TRUELAT2: A real value specifying, for ARW, the second true latitude for the Lambert conformal conic projection. For all other projections, truelat2 is ignored. No default value.

19. STAND_LON: A real value specifying, for ARW, the longitude that is parallel with the y-axis in the Lambert conformal and polar stereographic projections. For the regular latitude-

longitude projection, this value gives the rotation about the earth's geographic poles. For NMM, stand_lon is ignored. No default value.

20. POLE_LAT : For the latitude-longitude projection for ARW, the latitude of the North Pole with respect to the computational latitude-longitude grid in which -90.0° latitude is at the bottom of a global domain, 90.0° latitude is at the top, and 180.0° longitude is at the center. Default value is 90.0.

21. POLE_LON : For the latitude-longitude projection for ARW, the longitude of the North Pole with respect to the computational lat/lon grid in which -90.0° latitude is at the bottom of a global domain, 90.0° latitude is at the top, and 180.0° longitude is at the center. Default value is 0.0.

22. GEOG_DATA_PATH: A character string giving the path, either relative or absolute, to the directory where the geographical data directories may be found. This path is the one to which rel_path specifications in the GEOGRID.TBL file are given in relation to. No default value.

Many of these parameters, although required for every run, remain unchanged in the demonstration of the process. But there are many parameters that need to be changed for every run. And the editing of namelist files based on information given by Machine Learning sub-system is done automatically using java I/O.

5.6.4 Calculating the Parameters

As mentioned before, WRF sub-system receives boundaries in latitudes and longitudes. But, some important configuration parameters required by the model should be given in 'lambert' values in general. To obtain the corresponding lambert values from the given latitudes and longitudes, several java methods are included.

In finding the grid dimensions, first the differences of latitudes and longitudes are converted into meters. After that, the resolution values (dx and dy) are used to calculate the number of grid units in both x and y directions.

Calculating the center of the grid, the ref_lat and ref_lon values is much straight forward. The center of a given boundary can be calculated using simple math.

5.6.5 Automating WRF Model Execution

As emphasized earlier, it is important to automate the execution of WRF model for the performance evaluation purposes. One run of WRF model consists of several steps in several stages. Each of them should be done in order, providing necessary configurations and data on top of a correctly set environment. The information about how the environment was set, the correct order of steps which were followed to run the model, the automation script and running the automation script are included in this sub-section.

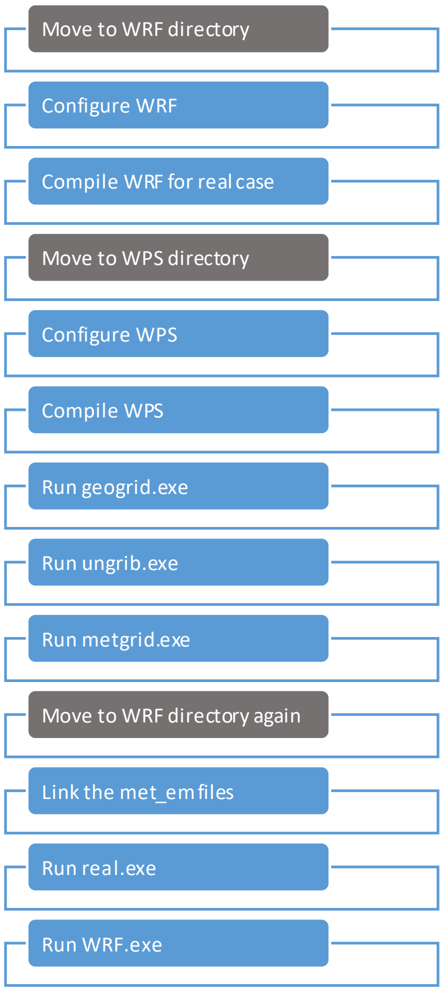
5.6.6 Setting the Environment

The WRF model requires UNIX environment to run. In addition, the following requirements should also be fulfilled.

- Fortran 90 or 95 and C compiler
- Perl 5.04
- netCDF
- HDF5
- Zlib

5.6.7 Steps and Order of Steps of Running WRF Model

Figure xx is an illustration of the steps that need to be followed in order to run the WRF model. To run each of these steps in the correct order, a script file is used. Following is a sample of the script file with real commands and data, which is used for the demonstration. The first two steps are omitted in the script as they only need to be run once.



```
#!/bin/bash
#Automated runnig of WRF model
START=$(date +%s);
#set paths to following directories
WRF_PATH="/home/chamil/WRFV3";
WPS_PATH="/home/chamil/WPS";
WRF_EMREAL_PATH="/home/chamil/WRFV3/test/em_real";
#set path to data
WRF_DATA="/home/chamil/WPS/ungrib/Colorado/";
START=$(date +%s);

#WPS
echo "Moving to WPS directory";
cd $WPS_PATH;
echo "configuring WPS...";
./configure;
echo "compiling WPS..";
./compile;

#WPS geogrid
echo "Running geogrid.exe";
./geogrid.exe;

#WPS ungrib
echo "linking Vtable";
ln -sf ungrib/Variable_Tables/Vtable.GFS Vtable;
echo "linking data path";
./link_grib.csh $WRF_DATA;
echo "Running ungrib.exe";
./ungrib.exe;

#WPS metgrid
echo "Running WPS metgrid.exe";
./metgrid.exe;

 #WRF final
echo "moving to WRF em_real";
cd $WRF_EMREAL_PATH;
echo "linking intermediate files";
ln -sf $WPS_PATH/met_em.d0* .;
echo "running real.exe";
./real.exe;
echo "running WRF.exe";
./wrf.exe;
 END=$(date +%s);
DIFF=$(( $END - $START ));
 echo "";
```

Figure 4.20 - Automation script of WRF model

```
echo "WRF model completed successfully";
echo "runtime = $DIFF seconds";
```

The complete run time of WRF model is also calculated in this script. Running this script also prompts for some inputs. To provide them automatically, another script is used inside which, the above script is executed with parameters. Finally, the latter script is run through a java program which is included in the GUI.

One other important thing to mention is that the WRF model deals with huge amounts of data in practice and requires a lot of computational power for its execution. Therefore, the model usually runs in supercomputers like Stampede cluster in XSEDE resources. The users can run the model via Apache Airavata for better performance. But, since the demonstration of this project uses a smaller amount of data, running the model locally will be sufficient.

5. Performance Evaluation

The attempt of Project Laridae to evaluate the success of the proposed solution architecture and implementation was carried out using number of tests. The methodology used for testing, the results gained, the method of evaluating results and a discussion about the results are included in this section. Section 5.1 describes the experimental setup. Results of the performance evaluation is presented in Section 5.2.while Section 5.3 discusses the results showed in Section 5.2.

5.1 Experimental Setup

5.1.1 Input Data

As described in the Implementation section, the weather stream data that were used for the testing purposes were in GRIB format. Even though the architecture proposes to continuously analyze data coming in real time, we selected a data set which is suited to display the true importance and gain of the system.

The actual input data set used covers a geographical area in North America, bounded by latitudes 12.19, 61.279 and longitudes -49.38, -152.85. The data set is chosen so that a time period of an entire day, starting from 19th December, 2006 and ending from 20th December, 2006 is covered. A thunderstorm of 50kts had occurred in the area Decatur within this time period. In the dataset, there are GRIB files containing data in 3-hour gaps. The initial geographical coverage of data is shown in the following image.

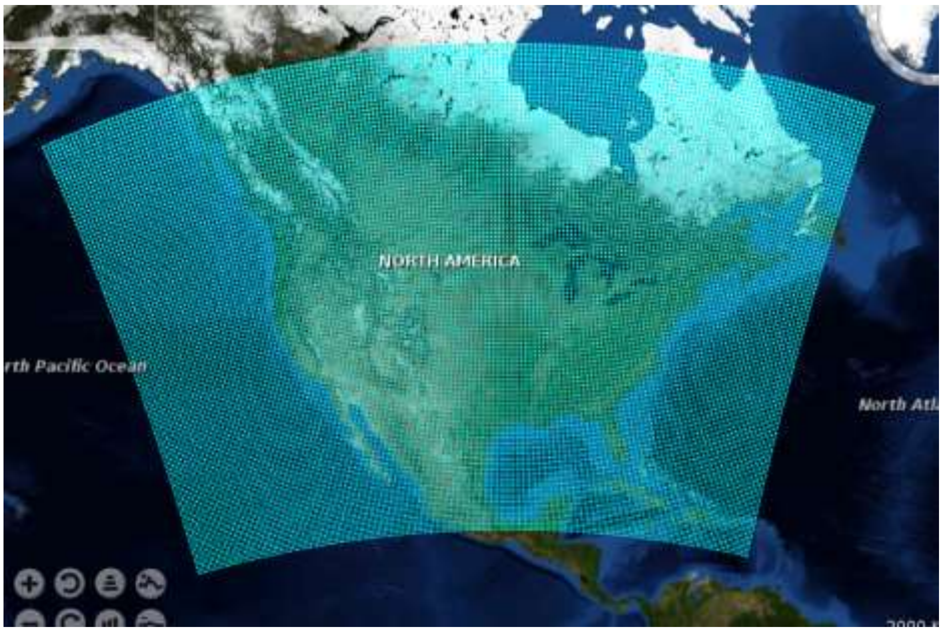


Figure 5.1- Initial geographical coverage of data

These data files are then programmatically decoded, or in domain specific words, 'degribbed' to obtain the data of the following four parameters corresponding to multiple data points:

- Best 4 layer lifted index
- Storm Relative Helicity
- Convective Inhibition

5.1.2 Obtaining Boundaries

The degribbed data are then fed to the Complex event processing sub-system as a stream. The CEP engine monitors the incoming weather data and for every 8 seconds, outputs a boundary in latitudes and longitudes. This boundary contains the marginal latitudes and longitudes of filtered data points. This is usually a smaller geographical area than the initial coverage of the input data. But depending on the distribution of the weather event, this boundary may still provide the initial coverage boundary itself. A sample output after the CEP processing is included below. This boundary is then fed to the Machine Learning system. There, clusters of data points are identified.

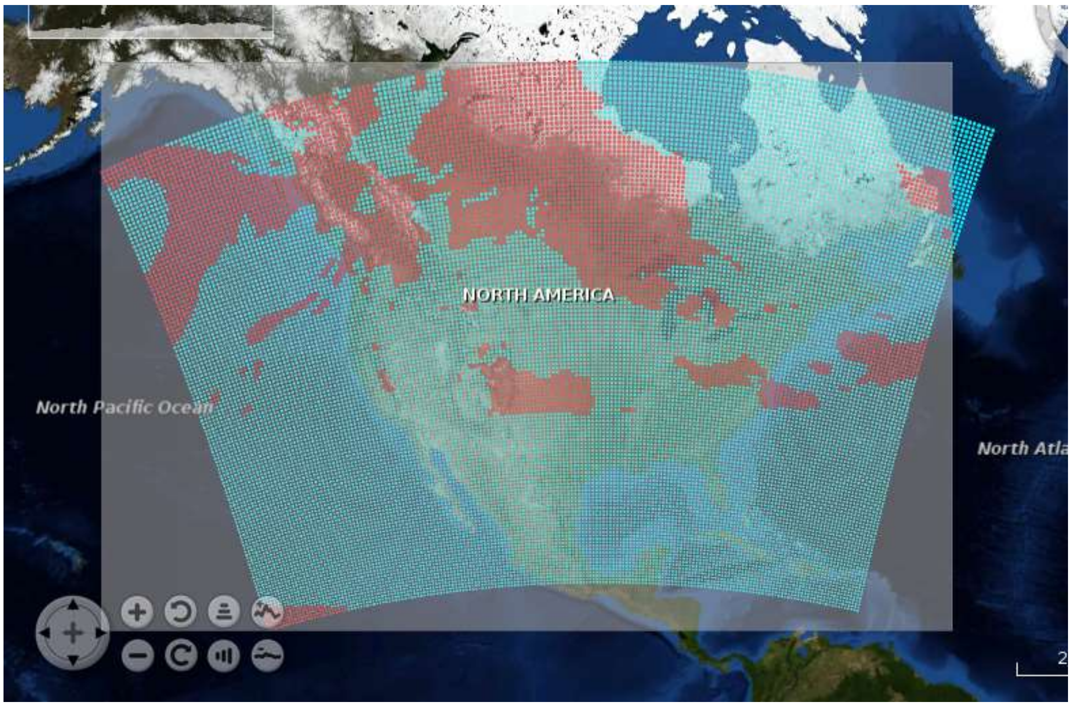


Figure 5.2 - Output from CEP sub-system

For clustering of data points, we use and compare two machine learning algorithms as described earlier. Sample output clusters for the above data set when using each of the Machine Learning algorithm are shown below.

Figure 5.3 displays the output of Optimized K-means algorithm.

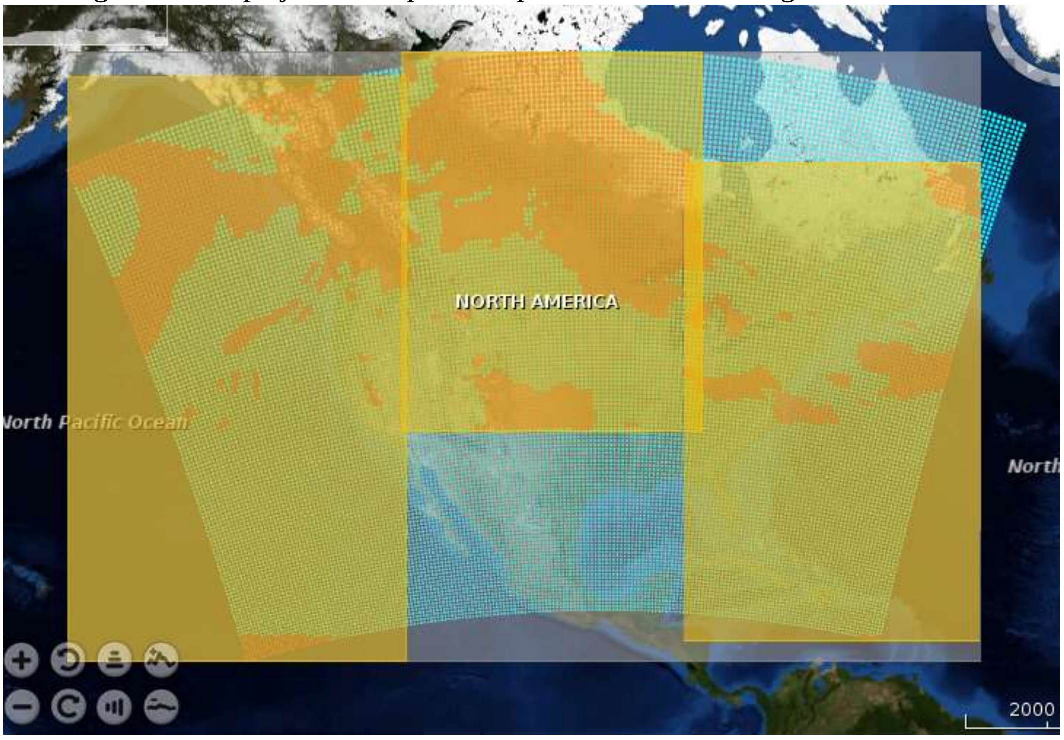


Figure 5. 3 - Output of ML sub-system when K-means is used

Figure 5.4 displays the results gained when the ML algorithm is changed to GMM.

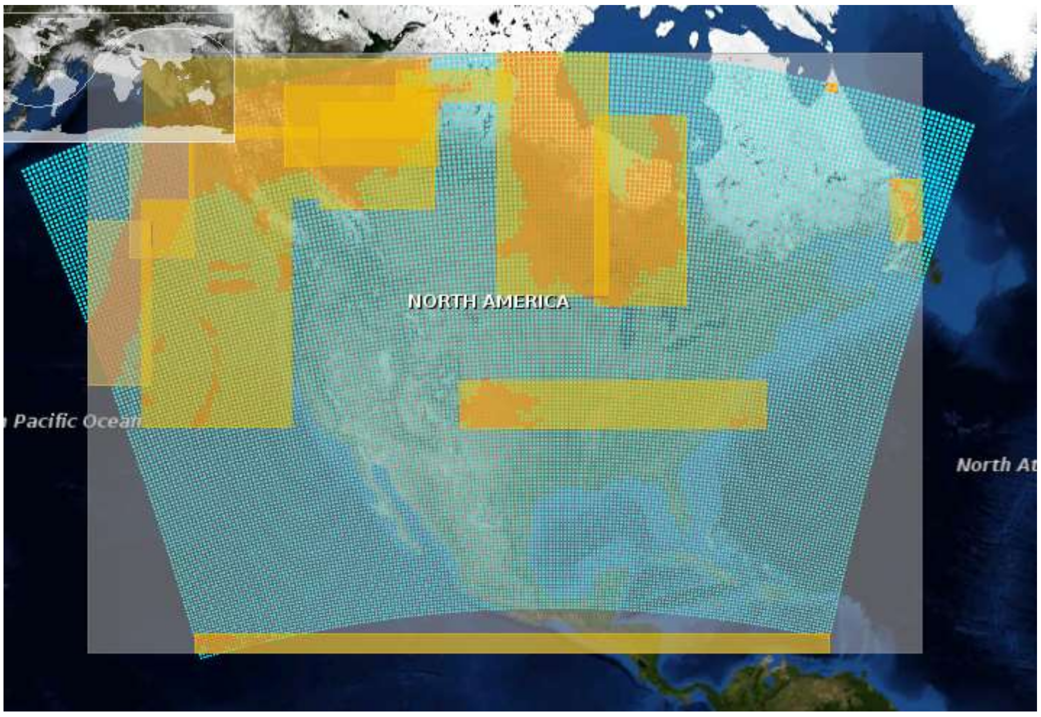


Figure 5.4 - Output of ML when GMM is used

5.1.3 Calculating Execution Time

After obtaining cluster boundaries for each Machine Learning algorithm one at a time, the WRF model is run for each cluster concurrently. As described earlier, the system is programmed to calculate the running time of each run. Therefore, it is possible to get the execution time of WRF model for each run. So, the aggregate runtime for a complete cycle of the system including Complex vent Processing, Machine Learning based processing and WRF model execution can be obtained. Since the data of from the clusters are processed concurrently in WRF model, the longest execution time result corresponds to the largest cluster.

Separately from this process, we also need to get the run-time of the WRF model for initial data set without the pre-processing of CEP and ML sub-systems. In other words, it is required to get the execution time of WRF model for the initial data set without limiting the concern to smaller areas but taking the entire area as one cluster. This can be easily done by running the WRF model without the intervention of the system.

After all this, the comparison of results was done.

5.2 Results

For the ease of reference, we use the following scenarios as named below:

Scenario 1:
Running the WRF model with all the initial data without any pre-processing or reduction of coverage

Scenario 2:
Running the system with CEP and ML pre-processing taking **K-means** as the Machine Learning algorithm.

Scenario 3:
Running the system with CEP and ML pre-processing taking **Gaussian Mixture Model** as the Machine Learning algorithm.

- *For Scenario 1*:

The total run time for execution	: **745 seconds.**

- *For Scenario 2*:

Execution time for CEP	: **20.735 seconds**
Execution time for ML	: **19.299 seconds**
Execution time of WRF model for each cluster:	
Cluster 1	: **165.632 seconds**
Cluster 2	: **136.175 seconds**

Cluster 3	: **127.816 seconds**
Cluster 4	: **39.147 seconds**

As the model executes parallel, the longest execution time:

20.735s + 19.299s + 165.632s = **205 seconds**

- *For Scenario 3:*

Execution time for CEP	: **15.75 seconds**
Execution time for ML	: **63.063 seconds**

Execution time of WRF model for each cluster:

Cluster 1	: **24.1 seconds**
Cluster 2	: **43.574 seconds**
Cluster 3	: **21.305 seconds**
Cluster 4	: **28.729 seconds**
Cluster 5	: **26.375 seconds**
Cluster 6	: **24.713 seconds**
Cluster 7	: **25.818 seconds**
Cluster 8	: **28.82 seconds**
Cluster 9	: **40.082 seconds**
Cluster 10	: **73.978 seconds**
Cluster 11	: **28.375 seconds**
Cluster 12	: **34.464 seconds**
Cluster 13	: **36.872 seconds**

As the model executes parallel, the longest execution time:

15.75s + 63.063s + 73.978s = **153 seconds**

The above results are graphed below so that they can be compared easily.

Below is the comparison of execution time when the WRF model is run with the initial data set with entire coverage VS the execution time when the data is pre-processed, using the K-means algorithm as the Machine Learning algorithm.

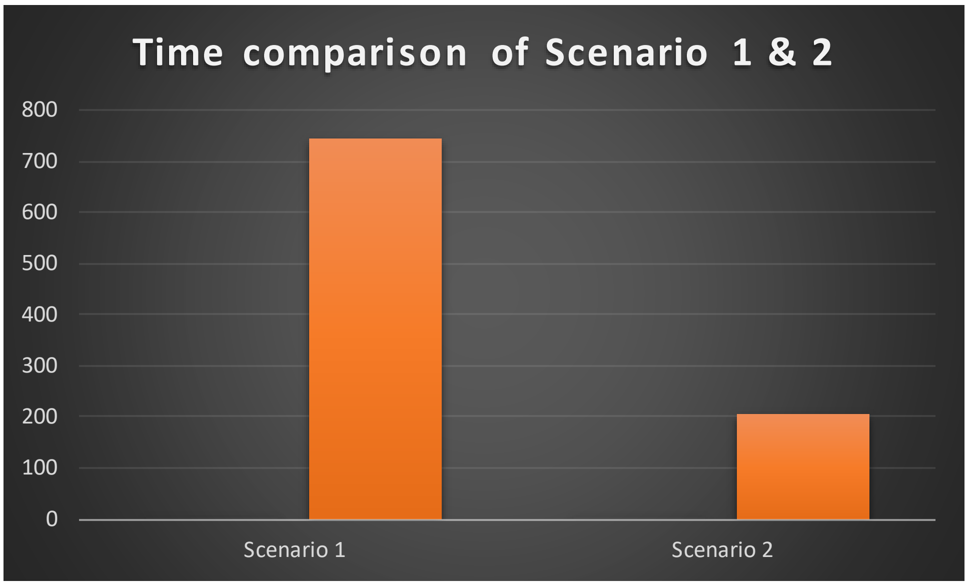


Figure 5.5 - Comparison of scenario 1 and 2

By this graph, it is clear that there is a significant time gain when the data are pre-processed. Below is a graph showing the time comparison when the Machine Learning algorithm used for pre-processing is changed to Gaussian Mixture Model.

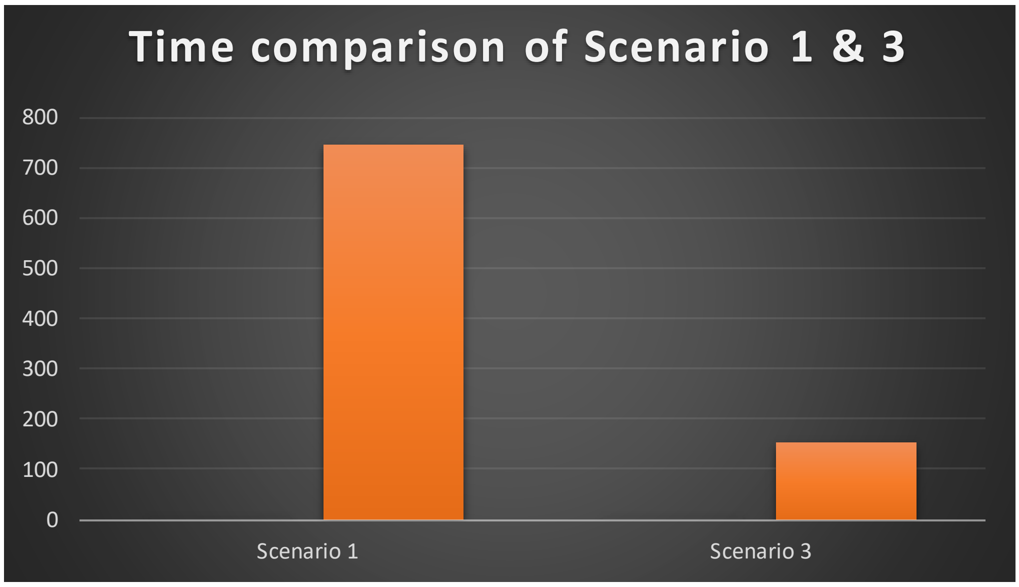


Figure 5.6 - Comparison of scenario 1 and 3

This graph also suggests that there is a notable time gain when the proposed solution architecture is followed. Finally, below is a graph which compares the time consumption when the two algorithms are used as the Machine Learning algorithm.

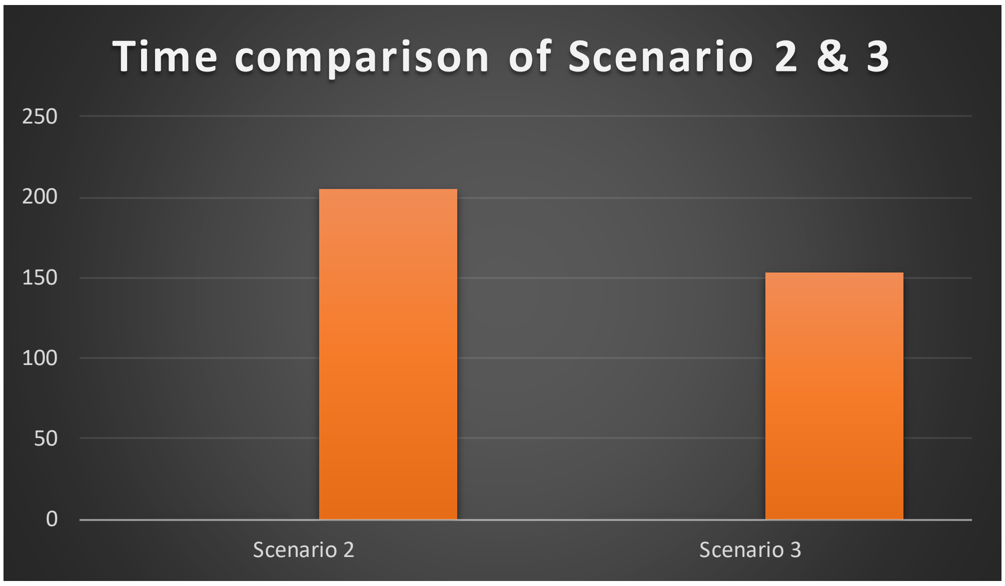


Figure 5.7 - Comparison of scenario 2 and 3

From this graph, we can come to the conclusion that the Gaussian Mixture Model algorithm is more efficient than K-means algorithm for the use of this project scope.

The time gain obtained by pre-processing data is not the only parameter to measure the success of the solution we propose. The accuracy of the forecasts should also be preserved when the processing is localized. To measure this, we graph the output generated from the WRF model with the IDV software mentioned above and show that the outputs of each cluster matches with the output of corresponding areas in Scenario 1. Relevant parameters should be chosen in order to show this.

Following are examples of similarity of outputs when the entire data coverage is considered VS when the smaller clusters are processed.

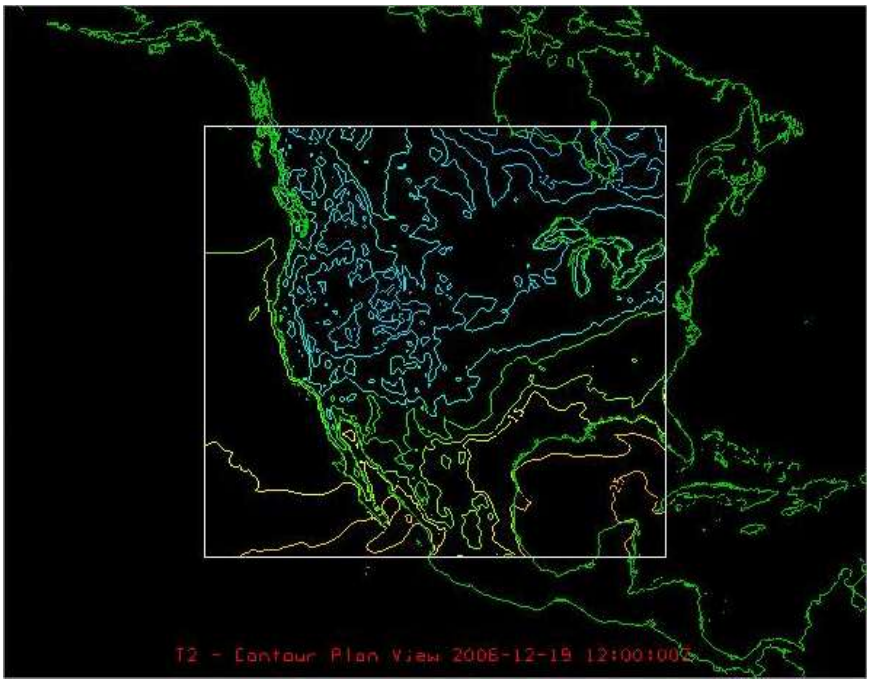


Figure 5.9 - WRF output without preprocessing

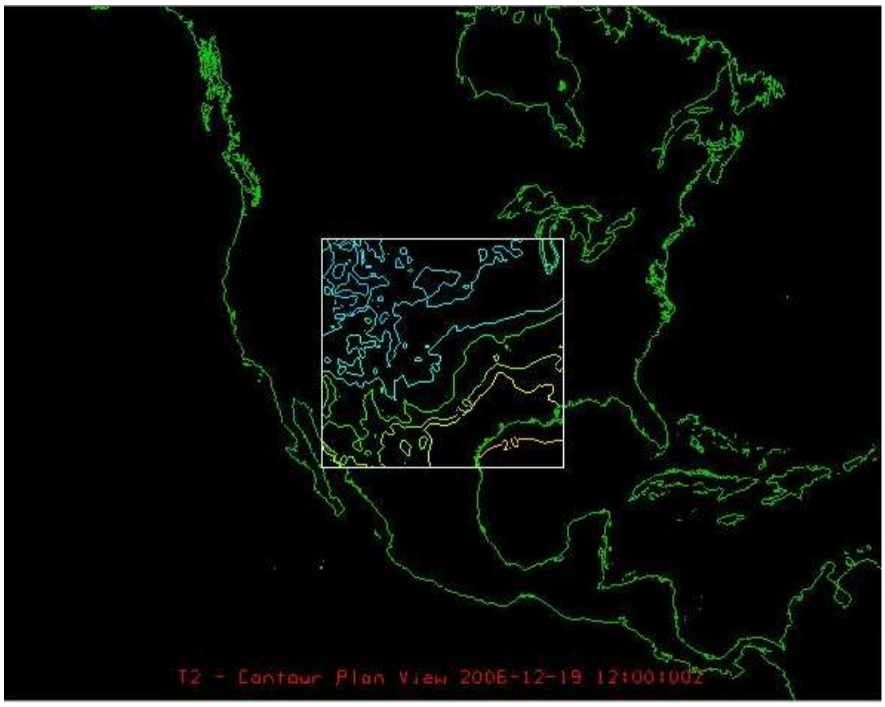


Figure 5.8 - WRF output for a cluster

Comparing the above two images, we can see that the forecast result of the localized run has preserved the accuracy of the initial forecast.

5.3 Discussion

Once the above results are analyzed, it is possible to claim that the proposed solution architecture regarding the stated problem domain has been successful in obtaining a significant time gain, preserving the accuracy in forecasting weather events. In addition, the comparison between two machine learning algorithms suggests that the Gaussian Elimination Algorithm has been more successful in clustering data points based on geographical distribution.

Using this time gain, the next improvement that can be applied to the system is to increase the resolution of data gathering. Data coming from a higher resolution can always give more accurate and reliable results. But, since the WRF model usually processes data from larger areas, increasing the resolution had not been an option because that way, more and more computational power and time will be required for the model to execute. But, because of the localization of concerns, increasing the resolution of data in areas identified as interesting would not affect the time and resource consumption like above. This is a major advantage of this solution architecture.

In implementing this solution architecture in practice, it would be inefficient to run the WRF model in a local computer because the amount of data to be processed is too large and requires higher computing power. Therefore, it is better to use a WRF server hosted in computers with high computational power, like the XSCEDE supercomputer clusters. In connecting to such resources, the suited way is to use a science gateway like Apache Airavata. But, since we use a smaller dataset to obtain test results, all the execution took place in local computers with average computational power.

In addition, concurrent runs of WRF model can also be easily handled if the above kind of approach is used. This is because it is possible to submit all the runs as multiple jobs. But, in obtaining results, WRF runs corresponding to each cluster were run serially and the time is calculated separately.

All in all, the reduction of time and resource consumption by localizing the areas of interest is a successful achievement in a computer science view while the opportunity it has opened to increase the resolution of data is also a successful achievement in a weather point of view.

6. Challenges Faced

In doing this project, there were many challenges in each of the requirement specification, design, implementation phases.

In the requirements specification phase, the most severe challenge we faced was the lack of domain knowledge of the group members. To better understand the project, we needed to understand concepts related to weather. Since we selected thunderstorms as the weather event to show case this concept, we started to overcome this challenge by researching about thunderstorms first. Then we expanded our knowledge gradually to learn about radar images, coordinate systems of earth and so on.

In the design phase, the biggest challenge we faced was the lack of documentation available to understand, set up and run the WRF model with proper configurations. It took a lot of effort and tests to finally understand the correct way to run the model and to correctly set up the running environment.

When it comes to implementation, the first Machine Learning algorithm we used was the K-means algorithm but it did not give the most optimized results at first. We researched about more algorithms and implemented the prototype to test the GMM algorithm too, which gave better clusters and in the end, lead to better performance results.

Interpreting the results gained from the WRF model too requires domain knowledge. This was a bigger challenge for us and after researching thoroughly and communicating with the domain experts, we obtained a solution, which is to show the similarity of the outputs corresponding to some particular parameters.

7. Future Work

The solution architecture that we discussed throughout this report mainly consisted of 3 components which are CEP layer, ML layer and complex weather algorithms layer. The collaboration of these three layers makes a profound solution in terms of computer science point of view since the 3 – layered architecture reduces the execution time by approximately 75% and reduces the required computational resources due to the early filtering of data.

But, to make this concept a completer success in meteorological point of view too, there need to be several other steps implemented. One such step is to increase the resolution of data collection based on the localized boundaries obtained through various levels of the system. This should be done by feeding the information back to the hardware which produces data streams so that they can adjust the frequency and resolution of data collection.

Another future improvement is to implement the system to use Apache Airavata, in order to remotely run WRF model by submitting jobs to the servers. This will be useful in handling huge amounts of data. To process huge amount of data, powerful servers are suitable and to communicate between the local machines where pre-processing systems reside and the powerful servers, Apache Airavata can be used.

8. Summary

Weather forecasts are crucial for day-to day life of many. The forecasts should be timely and accurate if they are to be useful. But, usually, since the weather forecasting models have to process huge volumes of data, it is a very time and resource consuming process and if the time should be reduced, the resolution of gathering data should be reduced which leads to a lower accuracy of forecasts.

To overcome these problems, Project Laridae proposes a solution architecture in which, all the data related to a particular region is not processed at once but goes through a pre-processing stage where the most likely areas of selected weather events are identified. The processing is then limited to those identified areas only, which reduces time and resource consumption while preserving the accuracy. In fact, by this method, data can be obtained in a higher resolution from those areas without over spending time and resources.

The architecture consists of a Complex Event processing sub-system, a Machine Learning sub-system and the WRF model. The performance evaluations are done by obtaining running times of WRF model in usual manner without any localization and with localization by identifying clusters. The use of two different Machine Learning algorithms are also compared. The results were good enough to successfully show the claimed purposes of the project and are discussed in a separate sub-section. Practically, this would be a good achievement in the domain of weather analytics.

References

[1] L. Kimball, "Complexity and the Mystery of Predicting the Weather", (Plexux Institute), [online] 2011, http://www.plexusinstitute.org/blogpost/656763/130737/Complexity-and the-Mysteryof-Predicting-the-Weather [Accessed: 15 July 2015].

[2] B. Plale, D. Gannon, J. Brotzge, K. Droegemeier, J. Kurose, D. McLaughlin, R. Wilhelmson, S. Graves, and M. Ramamurthy, "CASA and LEAD: Adaptive Cyberinfrastructure for Real-Time Multiscale Weather Forecasting," IEEE Computer Society, 2006.

[3] I. L. Narangoda, S. Suhothayan, S. Chaturanga, K. Gajasinghe, S. Perera, and V. Nanayakkara, "Siddhi: A Second Look at Complex Event Processing," 2011.

[4] L. Li, Z. Ma, L. Liu, and Y. Fan, "Hadoop-based ARIMA Algorithm and its Application in Weather Forecast," International Journal of Database Theory and Application, vol. 6, no. 5, 2013, pp. 119-132, doi:10.14257/ijdta.2013.6.5.11.

[5] F. Molteni et al., "The ECMWF Ensemble Prediction System: Methodology and validation", Quarterly Journal of the Royal Meteorological Society, vol. 122, no.529 , pp. 73-119, Jan. 1996, doi:10.1002/qj.49712252905.

[6] B. Pale et al., "Towards Dynamically Adaptive Weather Analysis and Forecasting in LEAD" 5th International Conference, Atlanta, GA, USA, May 2005, pp. 624-631, doi: 10.1007/11428848_81.

[7] "WSO2 Complex Event Processor", WSO2, [Online]. Available: http://wso2.com/products/complex-event-processor/. [Accessed July 2015].

[8] "Spark, Lightning-fast cluster computing", (Apache Software Foundation), [Online], https://spark.apache.org [Accessed: 8 June 2015].

[9] S. Marru et al., "Apache Airavata: A framework for Distributed Applications and Computational Workflows," SC 2011 Workshop on Gateway Computing Environments: Seattle, WA, USA, pp. 21-28, doi:10.1145/2110486.2110490.

[10] NOAA National Severe Storms Laboratory, "Thunderstorm Basics", 2015. [Online]. Available: http://www.nssl.noaa.gov/education/svrwx101/thunderstorms/. [Accessed: 03- Sep- 2015].

[11] W. Skamarock et al., "A description of the advanced research WRF version 2", No. NCAR/TN-468+ STR, National Center for Atmospheric Research Boulder Co Mesoscale and Microscale Meteorology Div, 2005.

[12] W. Wang et al., "User's Guide for Advanced Research WRF (ARW) Modeling System Version 3" (2011).

[13] J. Dudhia, “WRF Modeling System Overview”, [online], http://www2.mmm.ucar.edu/wrf/users/tutorial/201201/WRF_Overview_Dudhia.ppt.pdf, (Accessed: 24 June 2015).

[14] C. Hill et al., “The architecture of the earth system modeling framework”, Computing in Science and Engineering, vol. 6, no. 1, pp. 18 - 28, Jan-Feb 2004, 10.1109/MCISE.2004.1255817.

[15] B. Plale, D. Gannon, J. Brotzge, K. Droegemeier, J. Kurose, D. McLaughlin, R. Wilhelmson, S. Graves, and M. Ramamurthy, “CASA and LEAD: Adaptive Cyberinfrastructure for Real-Time Multiscale Weather Forecasting”, IEEE Computer Society, 2006.

[16] “Complex event processing”, Wikipedia, 2015. [Online]. Available: https://en.wikipedia.org/wiki/Complex_event_processing. [Accessed July 2015].

[17] “EsperTech - Products – Esper”, Espertech, 2006. [Online]. Available: http://www.espertech.com/products/esper.php [Accessed July 2015].

[18] “Sybase Aleri Streaming Platform”, Sybase, Inc, 2010.

[19] S. Vidich and J. Morell, “Complex Event Processing with Coral8,” Microsoft Corporation, 2008.

[20] M. R. Mendes, P. Bizarro, and P. Marques, “A Performance Study of Event Processing Systems,” Berlin, 2009.

[21] “Storm, distributed and fault-tolerant realtime computation”, 2015. [Online]. Available: https://storm.apache.org/. [Accessed: 07- Aug- 2015].

[22] Hortonworks, “Apache Storm”, 2014. [Online]. Available: http://hortonworks.com/hadoop/storm/. [Accessed: 07- Aug- 2015].

[23] “Welcome to Apache™ Hadoop!,” (Apache Software Foundation), [online] 2015, https://hadoop.apache.org [Accessed: 15 July 2015].

[24] “Apache Mesos”, (Apache Software Foundation), [online] 2014, http://mesos.apache.org [Accessed: 15 July 2015].

[25] “Apache Hadoop Next Generatoin MapReduce (YARN)”, (Apache Software Foundation), [online]2015, http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html [Accessed: 15 July 2015].

[26] “Spark Streaming”, (Apache Software Foundation), [online], https://spark.apache.org/streaming/ [Accessed: 15 July 2015].

[27] “Spark SQL”, (Apache Software Foundation), [online], https://spark.apache.org/sql. [Accessed: 15 July 2015].

[28] “Spark MLib”, (Apache Software Foundation), [online], https://spark.apache.org/mllib/sql [Accessed: 15 July 2015].

[29] “Spark GraphX,” (Apache Software Foundation), [online], https://spark.apache.org/graphx [Accessed: 15 July 2015].

[30] “Cassandra,” (Apache Software Foundation), [online] 2015, http://cassandra.apache.org [Accessed: 15 July 2015].

[31] M. Olson, “Map Reduce and Spark”, (Cloudera), [online] 2013, http://vision.cloudera.com/mapreduce-spark [Accessed: 15 July 2015].

[32] “Map Reduce,” (Wikipedia), [online] 2015, https://en.wikipedia.org/wiki/MapReduce [Accessed: 15 July 2015].

[33] “Apache Storm,” (Apache Software Foundation), [online] 2014, https://storm.apache.org [Accessed: 15 July 2015].

[34] “Apache Airavata,” [online] 2015, https://airavata.apache.org/ [Accessed: 23 June 2015].

[35] “The Weather Research & Forecasting Model”, 2015. [Online]. Available: http://www.wrf-model.org/index.php. [Accessed: 12- Jul- 2015].

[36] “XSEDE,” 2015. [Online]. Available: https://www.xsede.org/. [Accessed: 23 July 2015].

[37] M. Pierce et al., “Apache Airavata: Building Gateways to Innovation,” ApacheCon 2013, Portland, United States, Feb. 27, 2013.

[38] A. Craig, Topic: “Getting Started With High Performance Computing for Humanities, Arts, and Social Science,” Apr. 1, 2014.

[39] J. Towns et al., “XSEDE: Accelerating Scientific Discovery,” Computing in Science & Engineering, vol.16, no. 5, pp. 62-74, Sept.-Oct. 2014, doi:10.1109/MCSE.2014.80.

[40] Hugo, “K-means clustering,” [Online]. Available: http://www.onmyphd.com/?p=k means.clustering&ckattempt=1, [Accessed: 05-Jan-2016].

[41] D. T. Pham et al., “Selection of K in K-means Clustering,” Manufacturing Engineering Centre, Cardiff University, Cardiff, UK, DOI: 10.1243/095440605X8298.

[42] “WRF Online Tutorial,” 2016. [Online].Available: http://www2.mmm.ucar.edu/wrf/OnLineTutorial/ [Accessed: 05-Jan-2016].

[43] J. Grieser, “Convection Parameters”, June. 26, 2012.

[44] “Stability Indices”, [Online]. Available: http://weather.cod.edu/sirvatka/si.html [Accessed: 05-Jan-2016].

[45] Nomads.ncdc.noaa.gov, "Index of /data/meso-eta-hi/201511", 2016. [Online]. Available: http://nomads.ncdc.noaa.gov/data/meso-eta-hi/201511/. [Accessed: 12- Dec-2015]

[46] A. Taylor, "Degrib: NDFD GRIB2 Decoder", *Nws.noaa.gov*, 2016. [Online]. Available: http://www.nws.noaa.gov/mdl/NDFD_GRIB2Decoder/. [Accessed: 05- Jan- 2016].

APPENDIX

I. Implementation of the K-Means Algorithm. Apache Spark was used for this.

```
public static int findK(JavaRDD<Vector> points, int iterations,
      int runs){
      int kMax = (int)Math.sqrt(points.count());
      KMeansModel model = null;
      double Fk = 0;
      double ak = 0;
      double ak_1 = 0; //a(k - 1)
      double Sk = 0;
      double Sk_1 = 0; //S(k - 1)

      int dimensions = points.first().size();

      for (int k = 1;  k < kMax; k++) {
        KMeans kmeans = new KMeans();
        kmeans =
            kmeans.setEpsilon(epsilon).setK(k)
            .setMaxIterations(iterations).setRuns(runs)
            .setInitializationMode(KMeans.K_MEANS_PARALLEL());

        model = kmeans.run(points.rdd());
        JavaRDD<Integer> values = model.predict(points);
        Sk = findSk(k, model, values, points);
        ak = findak(k, dimensions, ak_1);
        Fk = findFk(k, Sk, Sk_1, ak);

        if(Fk < 0.85){
            return k;
        }
        Sk_1 = Sk;
        ak_1 = ak;
      }
      return 1;
}

public static double findSk(int k, KMeansModel model,
      JavaRDD<Integer> values, JavaRDD<Vector> points){
      if(k==1){
        return 0.0;
      }else {
        Vector[] centers = model.clusterCenters();
        JavaPairRDD<Integer, Vector> pointCenters =
            values.mapToPair(new PairFunction<Integer, Integer,
            Vector>() {
                  int x = 0;
            @Override
            public Tuple2<Integer, Vector> call(Integer i) throws
                  Exception {
                  return new Tuple2<>(x++, centers[i]);
            }
        });
        JavaPairRDD<Integer, Vector> indexedPoints =
            points.mapToPair(new PairFunction<Vector, Integer,
            Vector>() {
                    int x = 0;
                @Override
                    public Tuple2<Integer, Vector> call(Vector vector)
                        throws Exception {
                        return new Tuple2<Integer, Vector>(x++,
                            vector);
                    }
                });
            JavaPairRDD<Integer, Vector> union =
                pointCenters.union(indexedPoints);
            JavaPairRDD<Integer, Vector> indexedDistances =
                union.reduceByKey(new Function2<Vector, Vector, Vector>()
                {
                @Override
                    public Vector call(Vector vector, Vector vector2)
                        throws Exception {
                        return Vectors.dense(Vectors.sqdist(vector,
                            vector2));
                    }
                });
            JavaRDD<Double> distances = indexedDistances.map(new
                Function<Tuple2<Integer, Vector>, Double>() {
                @Override
                public Double call(Tuple2<Integer, Vector> tuple) throws
                    Exception {
                    return tuple._2().apply(0);
                }
            });
            return distances.reduce(new Function2<Double, Double,
                Double>() {
                @Override
                public Double call(Double a, Double b) throws Exception {
                    return a + b;
                }
            });
        }
    }

    public static double findak(int k, int Nd, double ak_1/*a(k-1)*/){
        if(k==1){
            return 1.0;
        }else if(k==2){
            return  1.0-3.0/(4.0*Nd);
        }else{
            return ak_1 + (1.0-ak_1)/6.0;
        }
    }
    public static double findFk(int k, double Sk,double Sk_1, double ak){
        if(k==1){
            return 1;
        }else if(Sk_1==0){
            return 1;
        }else{
            return Sk/(ak*Sk_1);
        }
    }
```

II. Implementation of GMM Algorithm using Apache Spark.

```
public static int findK(JavaRDD<Vector> points){
        int bestk = 0;
        double bestBIC = Double.MAX_VALUE;
        int kMax = 15;
        ArrayList<Double> BICs = new ArrayList<>();
        for (int k = 1;  k < kMax; k++) {
            double bic = ProbSum(points, k);
            BICs.add(bic);
            if(bic<bestBIC){
                bestBIC = bic;
                bestk = k;
            }
        }
        return bestk;
    }
}
public static double ProbSum(JavaRDD<Vector> points, int k){
        GaussianMixtureModel gmm = new
            GaussianMixture().setK(k).run(points.rdd());
        JavaRDD<double[]> probabilities =
            gmm.predictSoft(points.rdd()).toJavaRDD();
        JavaRDD<Double> maxProbabilities = probabilities.map(
            new Function<double[], Double>() {
            Double max;
            @Override
            public Double call(double[] probabilities) throws Exception {
                  max = 0.0;
                  for (double probability : probabilities) {
                    if (max < probability) {
                        max = probability;
                    }
                  }
                  return max;
            }
        });

        Double sum = maxProbabilities.reduce((Function2<Double, Double,
            Double>) (a, b) -> a+b);
        return sum;
}
```

III. Sample namelist.input file to configure WRF for execution

```
&time_control
 run_days                            = 0,
 run_hours                           = 12,
 run_minutes                         = 0,
 run_seconds                         = 0,
 start_year                          = 2000, 2000, 2000,
 start_month                         = 01,   01,   01,
 start_day                           = 24,   24,   24,
 start_hour                          = 12,   12,   12,
 start_minute                        = 00,   00,   00,
 start_second                        = 00,   00,   00,
 end_year                            = 2000, 2000, 2000,
 end_month                           = 01,   01,   01,
 end_day                             = 25,   25,   25,
 end_hour                            = 12,   12,   12,
 end_minute                          = 00,   00,   00,
 end_second                          = 00,   00,   00,
 interval_seconds                    = 21600
 input_from_file                     = .true.,.true.,.true.,
 history_interval                    = 180,  60,   60,
 frames_per_outfile                  = 1000, 1000, 1000,
 restart                             = .false.,
 restart_interval                    = 5000,
 io_form_history                     = 2
 io_form_restart                     = 2
 io_form_input                       = 2
 io_form_boundary                    = 2
 debug_level                         = 0
 /

 &domains
 time_step                           = 180,
 time_step_fract_num                 = 0,
 time_step_fract_den                 = 1,
 max_dom                             = 1,
 e_we                                = 74,    112,   94,
 e_sn                                = 61,    97,    91,
 e_vert                              = 30,    30,    30,
 p_top_requested                     = 5000,
 num_metgrid_levels                  = 27,
 num_metgrid_soil_levels             = 4,
 dx                                  = 30000, 10000,  3333.33,
 dy                                  = 30000, 10000,  3333.33,
 grid_id                             = 1,     2,     3,
 parent_id                           = 0,     1,     2,
 i_parent_start                      = 1,     31,    30,
 j_parent_start                      = 1,     17,    30,
 parent_grid_ratio                   = 1,     3,     3,
 parent_time_step_ratio              = 1,     3,     3,
 feedback                            = 1,
 smooth_option                       = 0
 /

 &physics
 mp_physics                          = 3,     3,     3,
 ra_lw_physics                       = 1,     1,     1,
 ra_sw_physics                       = 1,     1,     1,
 radt                                = 30,    30,    30,
 sf_sfclay_physics                    = 1,      1,      1,
 sf_surface_physics                   = 2,      2,      2,
 bl_pbl_physics                       = 1,      1,      1,
 bldt                                 = 0,      0,      0,
 cu_physics                           = 1,      1,      0,
 cudt                                 = 5,      5,      5,
 isfflx                               = 1,
 ifsnow                               = 1,
 icloud                               = 1,
 surface_input_source                 = 1,
 num_soil_layers                      = 4,
 sf_urban_physics                     = 0,      0,      0,
 /

 &fdda
 /

 &dynamics
 w_damping                            = 0,
 diff_opt                             = 1,       1,       1,
 km_opt                               = 4,       4,       4,
 diff_6th_opt                         = 0,       0,       0,
 diff_6th_factor                      = 0.12,    0.12,    0.12,
 base_temp                            = 290.
 damp_opt                             = 0,
 zdamp                                = 5000.,  5000.,  5000.,
 dampcoef                             = 0.2,     0.2,     0.2
 khdif                                = 0,       0,       0,
 kvdif                                = 0,       0,       0,
 non_hydrostatic                      = .true., .true., .true.,
 moist_adv_opt                        = 1,       1,       1,
 scalar_adv_opt                       = 1,       1,       1,
 /

 &bdy_control
 spec_bdy_width                       = 5,
 spec_zone                            = 1,
 relax_zone                           = 4,
 specified                            = .true., .false.,.false.,
 nested                               = .false., .true., .true.,
 /

 &grib2
 /

 &namelist_quilt
 nio_tasks_per_group = 0,
 nio_groups = 1,
 /
```