Krishna iResearch Intelligent Cloud Platform

1.AstroInfer – set of python scripts and C++ code analyzes large datasets for patterns (numeric and text data)
2.USBmd – a debugging software for USB. In the long term might be implemented as a Linux Driver Verifier
3.Virtual Generic Os – VIRGO – a new linux kernel variant with Machine Learning and Cloud infrastructure capabilities added as part of Linux kernel itself. In other words VIRGO is a Cloud and Machine-Learning enabled Operating System Kernel than in userspace application layer.
4.KingCobra – a decentralized message queueing kernel module system using Linux kernel workqueue and native queues based on post-office REQUEST-REPLY protocol.
5.Acadpdrafts – Implementation for drafts and publications in https://sites.google.com/site/kuja27/

Thus VIRGO Linux is pivotal in the above that interacts with components in KingCobra, USBmd, AsFer and Acadpdrafts

Krishna iResearch Intelligent Cloud Platform Products Design

(https://sites.google.com/site/kuja27/, https://sourceforge.net/u/userid-769929/profile/, http://sourceforge.net/p/acadpdrafts/code/ci/master/tree/Krishna_iResearch_opensourceproducts_archdiagram.pdf)

```
#---------------------------------------------------------------------------------------------------`
#ASFER - a ruleminer which gets rules specific to a query and executes them (component of iCloud Platform)
#This program is free software: you can redistribute it and/or modify
#it under the terms of the GNU General Public License as published by
```

#Srinivasan Kannan (alias) Ka.Shrinivaasan (alias) Shrinivas Kannan
#Independent Open Source Developer, Researcher and Consultant
#Ph: 9789346927, 9003082186, 9791165980
#Open Source Products Profile(Krishna iResearch):
#http://sourceforge.net/users/ka_shrinivaasan
#https://www.ohloh.net/accounts/ka_shrinivaasan
#Personal website(research): https://sites.google.com/site/kuja27/
#emails: ka.shrinivaasan@gmail.com, shrinivas.kannan@gmail.com,
kashrinivaasan@live.com
#--

Copyright attributions for other open source products dependencies:

1.Maitreya's Dreams - http://www.saravali.de (some bugs were fixed locally in degree computation of textual display mode)
2.SVMLight - http://svmlight.joachims.org/
3.BioPython and ClustalOmega Multiple Sequence Alignment BioInformatics Tools (www.biopython.org, www.ebi.ac.uk/Tools/msa/clustalo/)

Open Source Design and Academic Research Notes have been uploaded to

http://sourceforge.net/projects/acadpdrafts/files/MiscellaneousOpenSourceDesignAndAcademicResearchNotes_2013-08-11.pdf/download

Presently a complete string sequence mining subsystem and classification algorithms with indexing have been implemented for mining patterns in rules and encoded strings and executing those rules (in special case,an encoded horoscope).

****

AstroInfer Classifiers - Documentation on the dataset files used

****

decisiontree-attrvalues.txt - Attributes based on which decision is done and the values they can take (comma separated list)
decisiontree-test.txt - Test dataset with set of values for attributes in each line
decisiontree-training.txt - Training dataset with set of values for attributes and class it belongs in each line
test-set.txt - List of article id(s) to be classified - NaiveBayes
topics.txt - List of classes the article id(s) in training set belong to - NaiveBayes
training-set.txt - List of articles id(s) already classified - training dataset for NaiveBayes
word-frequency.txt - Words and their frequencies of occurrence in all articles - NaiveBayes
words.txt - words, the article id(s) having those words and number of occurrences of those words within specific article id (~ separated)

Above XXX.txt files need to be populated after doing some preprocessing of the articles to be classified. Preprocessing might include finding the word frequencies in the articles, finding classes of the articles, finding attributes and possible values of those attributes. At present the asfer.enchoros file contains encoded horoscopes for single class of events. Thus classification is redundant. But if it has encoded horo strings for all events then to filter out strings of a particular class of events, classification is needed.

Python script for autogenerating above txt files has been added under python-src/autogen_classifier_dataset.This script needs to be changed forgenerating training and test set files.

****

DESIGN NOTES, THEORETICAL INTERLUDES(might have errors), TODO AND NICE TO HAVE FEATURES (long-term design goals with no deadline)

****

1a. Test with Massive Data Sets of encoded strings for pattern mining. Algorithms for Approximate Kolmogorov Complexity, Minimum Description Length, Compressed Sensing (used in Image and Signal Processing) which "compress" large data to give short reconstructible description.

1b. An experimental text compression algorithm that deletes vowels and stores only consonants as far as meaning is not altered. For example "follow" could be stored as "fllw". Since on an average every third letter in an english word is a vowel, approximate compression is 33%. Message is reconstructed using most probable meaningful word for "fllw" (For example "follow" could be more probable than "fellow" or vice versa). This is similar to Texting in phones and to some extent encoding in Match Rating (http://en.wikipedia.org/wiki/Match_rating_approach). One more example could be "Decision" which can be compressed as "Dcsn". An interesting phonetic aspect of this is that "Decision" is spelt as "Dee-C-shan", or each vowel following a consonant is subsumed or coalesced into the preceding consonant while spelling. Thus "Dcsn" gives 50% compression ratio.

2. Create Class Association Rules with Frequent Itemsets(This may not be as befitting as Sequence Alignment as string mining is needed and not
itemsets)

3. (DONE) Implementation of String Distance measure using different distance measures (python-src/StringMatch.py) for comparing Encoded Strings in addition to pairwise and Multiple Sequence Alignment.

4. Construct a Hidden Markov Model with State transitions and Observations(events). If the number of states are exponential then this will be
infeasible.

5. Correlate with Rules from Classics (BPHS, BJ etc.,) with the mined datasets for particular class of events (Special Case of Mundane Predictive Model is described in items 47 to 51 and basic framework is already implemented)

6. (DONE) Integrate Classifiers in above String Pattern mining (classify strings and then mine).

7. Add more decision making algorithms - Graphical Models , Bayesian Network, Belief Propagation and CRFs, Mining Patterns in Numerical Datasets (separate set of items described later)

8. (DONE) InterviewAlgorithm code in
https://sourceforge.net/p/asfer/code/146/tree/python-src/InterviewAlgorithm/InterviewAlgorithmWithIntrinisicMerit.py and Classification based on indegrees of wordnet subgraph node in action item 6 above (as mentioned in
http://arxiv.org/abs/1006.4458 ,
https://sites.google.com/site/kuja27/TAC2010papersubmission.pdf?attredirects=0 and
http://www.nist.gov/tac/publications/2010/participant.papers/CMI_IIT.proceedings.pdf).
Related Publication Drafts are:
https://sites.google.com/site/kuja27/LowerBoundsForMajorityVotingPseudorandomChoice.pdf?attredirects=0,
https://sites.google.com/site/kuja27/CircuitForComputingErrorProbabilityOfMajorityVoting.pdf?attredirects=0,
https://sites.google.com/site/kuja27/IndepthAnalysisOfVariantOfMajorityVotingwithZFAOC.

pdf?attredirects=0. Test C++ code written in 2006 for computing Majority voting error probability is at: http://sourceforge.net/p/asfer/code/HEAD/tree/cpp-src/miscellaneous/pgood.cpp and python script written in 2010 is at: http://sourceforge.net/p/asfer/code/HEAD/tree/python-src/pgood.py

9. Add Graph search (for example subgraph isomorphism, frequent subgraph mining etc.,) for "inferring" a graph from dataset with edges as relations and nodes as entities(astronomical or otherwise).Reference survey of FSM algorithms: http://cgi.csc.liv.ac.uk/~frans/PostScriptFiles/ker-jct-6-May-11.pdf

10. (DONE) Use of already implemented Bayesian and Decision Tree Classifiers on astronomical data set for prediction (this would complement the use of classifiers for string mining) - autogen_classifier_dataset/ directory contains Automated Classified Dataset and Encoded Horoscopes Generation code.

11. (DONE) Added Support for datasets other than USGS EQ dataset(parseUSGSdata.py). NOAA HURDAT2 dataset has been parsed and encoded pygen horoscope string dataset has been autogenerated by parseNOAA_HURDAT2_data.py

12. AstroPsychoAnalysis - Integration of Psychology stimulus tools like PsychoPy and correlating them with conclusions derived from AstroInfer for AstroPsychoProfiling of events and entities.

13. Implement prototypical PAC learning of Boolean Functions from dataset.

14. (ONGOING) Adapt or make AstroInfer subservient to VIRGO Platform which has a long term design goal of Approximately Intelligent Cloud Operating System Framework so that already implemented Kernelspace Intermodule Function Invocation functionality of VIRGO when integrated with AstroInfer adds power to cloud i.e Kernelspace invocation of the AstroInfer machine learning code running on top of these cloud hardware can drive the components of the cloud hardware to make the VIRGO platform as one Giant

Geographically Distributed Robotic OS that "learns" ,"decides" from datasets and "drives" using AstroInfer code. AsFer+USBmd+VIRGO integrated cloud platform with machine learning features will be henceforth known as"Krishna iResearch Intelligent Cloud Platform - iCloud".

15.Unsupervised Named Entity Recognition using Conditional Random Fields.

##

A. Design Notes on Mining Numerical Datasets

##

16. (DONE-Minimum Implementation) Period Three theorem (www.its.caltech.edu/~matilde/LiYorke.pdf) which is one of the earliest results in Chaotic NonLinear Systems, implies any data that has a periodicity of 3 can have larger periods. Fractals, Mandelbrot-Julia Sets have modelled natural phenomena. Thus finding if a data has periodicity or Chaos and if it has sensitive dependence on initial conditions (for example logistic equations similar to x(n+1) = kx(n)(1-x(n))) is a way to mine numerical data. Computation of Hausdorff-Besicovitch Dimension can be implemented to get Fractal Dimension of an uncountable set (this has to be visualized as a fractal curve than set of points). In addition to these, an Implementation of Chaotic PRG algorithms as described in https://sites.google.com/site/kuja27/ChaoticPRG.pdf?attredirects=0 and https://sites.google.com/site/kuja27/Analysis%20of%20a%20Randomized%20Space%20Filling%20Algorithm%20and%20its%20Linear%20Program%20Formulation.pdf?attredirects=0 can be done.

17. (DONE) Entropy of numerical data - Sum of -Pr(x)LogPr(x) for all possible outcome probabilities for a random variable. This is already computed in Decision Tree Classifier as impurity measure of a dataset.

===

==================

(*)18. (ONGOING) COMPLEMENT OF A FUNCTION - Approximating or Interpolating with a Polynomial: This is quite expensive and is undecidable for infinite sets. Better alternative is to approximate with Spline interpolant polynomials, for example, cubic splines which have less approximation error. (On a related note, algorithms described by the author (that is, myself) in http://arxiv.org/pdf/1106.4102v1.pdf for construction of a complement of a function are also in a way eliciting pattern in numerical data. The polynomial interpolation algorithm for complement finding can be improved with Spline interpolants which reduce the error and Discrete Fourier Transform in addition to Fourier series for the boolean expression. Adding this as a note here since TeX file for this arXiv submission got mysteriously deleted and the PDF in arXiv has to be updated somehow later.). A test python script written in 2011 while at CMI for implementing the above is at: http://sourceforge.net/p/asfer/code/HEAD/tree/python-src/complement.py. Also Trigonometric Polynomial Interpolation which is a special case of Polynomial interpolation, has following relation to DFT X(i)s:

p(t) = 1/N [X0 + X1*e^(2*pi*i*t) + ... upto XN] and

p(n/N) = xn

Thus DFT amd interpolation coincide in the above.

Theorems on prime number generating polynomials - special case is to generate all primes or integral complement of xy=z

----

Legendre - There is no rational algebraic function which always gives primes.
Goldbach - No polynomial with integer coefficients can give primes for all integer values
Jones-Sato-Wada-Wiens - Polynomial of degree 25 in 26 variables whose values are exactly primes exists

Polynomial interpolation and Fourier expansion of the boolean function in http://arxiv.org/pdf/1106.4102v1.pdf probably would have non-integral coefficients due to the above.

Additional references:

18.1 Google groups thread reference 2003 (OP done by self):
https://groups.google.com/forum/#!search/ka_shrinivaasan%7Csort:relevance
%7Cspell:false/sci.math/RqsDNc6SBdk/Lgc0wLiFhTMJ
18.2 Math StackExchange thread 2013:
http://math.stackexchange.com/questions/293383/complement-of-a-function-f-2n-n-in-mathbbn-0-n-rightarrow-n1
18.3 Theory of Negation - http://mentalmodels.princeton.edu/skhemlani/portfolio/negation-theory/ and a quoted excerpt from it :
"... The principle of negative meaning: negation is a function that takes a single argument, which is a set of fully explicit models of possibilities, and in its core meaning this function returns the complement of the set..."
18.4 Boolean Complementation [0 or 1 as range and domain] is a special case of Complement Function above (DeMorgan theorem -
http://www.ctp.bilkent.edu.tr/~yavuz/BOOLEEAN.html)
18.5 Interpolation - http://caig.cs.nctu.edu.tw/course/NM07S/slides/chap3_1.pdf
18.6 Formal Concept Analysis - http://en.wikipedia.org/wiki/Formal_concept_analysis,
http://ijcai.org/papers11/Papers/IJCAI11-227.pdf
18.7 Prime generating polynomials - http://en.wikipedia.org/wiki/Formula_for_primes
18.8 Patterns in primes - every even number > 2 is sum of 2 primes - Goldbach conjecture : http://en.wikipedia.org/wiki/Goldbach%27s_conjecture
18.9 Arbitrarily many Arithmetic progressions - Green-Tao theorem:
http://en.wikipedia.org/wiki/Green%E2%80%93Tao_theorem
18.10 Prime generating functions -
https://www.sonoma.edu/math/colloq/primes_sonoma_state_9_24_08.pdf

==

19. Approximating with some probability distribution - Gaussian, Binomial etc., that model the probability of occurence of a datapoint in the set

20. Streaming algorithms - Finding Frequency moments, Heavy Hitters-Hitting Sets(most prominent items), Distinct Elements etc., in the numerical dataset. Usually numerical data occur in streams making these best choice for mining numerical data.

21. Usual Probabilistic Measures of Mean, Median, Standard Deviation, Curve fitting on the numeric data.

22. (ONGOING - using R+rpy2) Application of Discrete Fourier Transform, Logistic Regression and Gradient Descent

23. Least Squares Method on datapoints y(i)s for some x(i)s such that f(x)~y needs to be found. Computation of L0, L1 and L2 norms.

24. K-Means and kNN Clustering - unsupervised and supervised clustering based on coordinates of the numerical dataset

25. (DONE) Discrete Hyperbolic Factorization - Author has been working on a factorization algorithm with elementary proof based on discretization of a hyperbola since 2000 and there seems to be some headway recently in 2013. To confirm the polylog correctness, a minimal implementation of Discrete Hyperbolic Factorization (and could be even less than polylog due to a weird upperbound obtained using stirling formula) has been added to AstroInfer repository at:
http://sourceforge.net/p/asfer/code/HEAD/tree/cpp-src/miscellaneous/DiscreteHyperbolicFactorizationUpperbound.cpp with factors output logs.

26. (DONE) Multiple versions of Discrete Hyperbolic Factorization algorithms have been uploaded as drafts in https://sites.google.com/site/kuja27:

1)

http://sourceforge.net/projects/acadpdrafts/files/DiscreteHyperbolicFactorization_UpperboundDerivedWithStirlingFormula_2013-09-10.pdf/download and

2) http://sourceforge.net/projects/acadpdrafts/files/DiscreteHyperbolicPolylogarithmicSieveForIntegerFactorization_updated_rectangular_interpolation_search_and_StirlingFormula_Upperbound.pdf/download(and multiple versions due to various possible algorithms for search and upperbound technique used)

27. (DONE) NC PRAM version of Discrete Hyperbolic Factorization:

1) An updated NC PRAM version of Discrete Hyperbolic Factorization has been uploaded at: http://sourceforge.net/projects/acadpdrafts/files/DiscreteHyperbolicPolylogarithmicSieveForIntegerFactorization_PRAM_TileMergeAndSearch_And_Stirling_Upperbound.pdf/download that does PRAM k-merge of discrete tiles in logarithmic time before binary search on merged tile.

2) Preliminary Design notes for CRCW PRAM implementation of the above is added to repository at: http://sourceforge.net/p/asfer/code/237/tree/ImplementationDesignNotesForDiscreteHyperbolicFactorizationInPRAM.jpg. This adds a tile_id to each tile so that during binary search, the factors are correctly found since coordinate info gets shuffled after k-tile merge.

28. (ONGOING) Above Discrete Hyperbolic Factorization can be used as a numerical dataset analysis technique. Discrete Hyperbolic Factorization uses only elementary geometric principles which can be fortified into an algebraic geometry result, because of strong connection between geometry (hyperbola) and algebra (unique factorization theorem) .

##

B. EXPERIMENTAL NON-STATISTICAL INFERENCE MODEL BASED ON ALREADY KNOWN THEORETICAL COMPUTER SCIENCE RESULTS:

##
##

28. (DONE) The classification using Interview Algorithm Definition Graph (obtained from Recursive Gloss Overlap algorithm described in http://arxiv.org/abs/1006.4458 , https://sites.google.com/site/kuja27/TAC2010papersubmission.pdf?attredirects=0 and http://www.nist.gov/tac/publications/2010/participant.papers/CMI_IIT.proceedings.pdf) wordnet node indegrees can classify a same document in multiple classes without any training set (unsupervised). Thus same document will be in multiple classes. This can be visualized as one stack per class and documents in same class are pushed into a stack corresponding to each class. When a same document is across multiple classes, a document can be visualized as a "hyperedge" of a hypergraph that transcends multiple class nodes. Here each class node in this non-planar graph (or multi-planar graph) is a stack.

29. (DONE) Thus if number of classes and documents are infinite, an infinite hypergraph is obtained. This is also somewhat a variant of an inverted index or a hashtable where in addition to hashing ,the chained buckets across the keys are interconnected (Quite similar to https://sites.google.com/site/kuja27/SurvivalIndexBasedTxnTimeoutManager.pdf?attredirects=0 and https://sites.google.com/site/kuja27/PhDThesisProposal.pdf?attredirects=0). This is also similar to "Connectionism" in Computational Psychology which allows cycles or interconnections in Perceptrons and Neural Networks. An old version of this was written in an old deleted blog few years ago in 2006 at: http://shrinivaskannan.blogspot.com. If the relation between Hash Functions and Integer Partitions is also applied as described using Generating Functions in https://sites.google.com/site/kuja27/IntegerPartitionAndHashFunctions.pdf?attredirects=0 then an upperbound on number of possible Hypergraphs (visualizing them as interconnected Hash tables) can be derived which is a stronger notion to prove.

30.Above multiplanar hypergraph is quite related as a theoretical construct to a Pushdown Automata for Context Free Grammar. Also the Definition Graph Construction using Recursive Gloss Overlap is a Context-Sensitive counterpart of Parse Trees obtained for Context Free Grammar, but ignoring Parts-Of-Speech Tagging and relying only on the relation between words or concepts within each sentence of the document which is mapped to a multipartite or general graph. Infact this Definition Graph and Concept Hypergraph constructed above from indegrees quite resemble a Functional Programming Paradigm where each relationship edge is a Functional Program subroutine and the vertices are the parameters to it. Thus a hyperedge could be visualized as Composition Operator of Functional Programs including FP routines for Parts-of-Speech if necessary [WordNet or above Concept Hypergraph can be strengthened with FPs like LISP, Haskell etc.,]. RecursiveNeuralNetworks(MV-RNN and RNTN) have a notion of compositionality quite similar to Functional Programming in FPs (Reference: http://nlp.stanford.edu/~socherr/EMNLP2013_RNTN.pdf) but for tree structures. Applying the compositionality for Concept Hypergraphs constructed above using FPs is a possible research direction.

31.But above is not context free because context is easily obtainable from the hyperedge which connects multiple classes. As an experimental gadget above seems to represent an alternative computational model for context sensitivity and also process of human thought. This needs HyperEdge Search algorithms for hypergraph. A Related Neuropsychological notion of Hippocampus Memory Allocation in Human Brain can be found in http://people.seas.harvard.edu/~valiant/Hippocampus.pdf. Also a similar gadget is a special case of Artificial Neural Network - http://en.wikipedia.org/wiki/Hopfield_network - Hopfield Network - used to model human associative memory (pattern based memory retrieval than address based retrieval). Hyperedges of the above hypergraph can be memory vectors retrieved in an associative memory.

32.An interesting academic question to pose is - "Does the above hypergraph have a strong connectivity and a diameter upperbounded by a constant?". Equivalently, is the collection of universal knowledge close-knit or does nature connect seemingly unrelated concepts beyond what is "observable"? For example, path algorithms for Hypergraphs etc., can be applied to check s-t connectivity of two concepts. Some realworld linear programming and optimization problems which have strong natural applications are KnapSack problem,Tiling Problem or Packing problem that occur in everyday life of humanbeings (packing items for travel, arranging on a limited space etc.,). Thus the concept hypergraph might have a path connecting these.

33.A recent result of Rubik's cube (http://www.cube20.org/) permutations to get solution having a constant upperbound (of approximately 20-25 moves) indicates this could be true (if all intermediary states of Rubik's transitions are considered as vertices of a convex polytope of Concepts then isn't this just a Simplex algorithm with constant upperbound? Is every concept reachable from any other within 20-25 moves? Or if the above hypergraph is considered as a variant of Rubik's Cube in higher dimensions, say "Rubik's Hypercube" then is every concept node reachable from the other within 20 logical implication moves or is the diameter upperbounded by 20?). Most problems for hypergraph are NP-Complete which are in P for graphs thus making lot of path related questions computationally harder. Counting the number of paths in hypergraph could be #P-Complete. There was also a recent study on constant upperbound for interconnectedness of World Wide Web document link graph which further substantiates it (http://www9.org/w9cdrom/160/160.html - paragraph on diameter of the Strongly Connected Component Core of WWW). Probably this is a special case of Ramsey theory that shows order emerging in large graphs (monochromatic subgraphs in an arbitrary coloring of large graph).

34.Mapping Rubik's Cube to Concept Wide Hypergraph above - Each configuration in Rubik's Cube transition function can be mapped to a class stack node in the Concept Hypergraph. In other words a feature vector uniquely identifies each node in each class stack vertex of the Hypergraph. Thus move in a Rubik's cube is nothing but the hyperedge or a part of the hyperedge that contains those unique vectors. Thus hyperedges signify the Rubik's cube moves. For example if each class stack vertex is denoted as c(i) for i-th class and each element of the stack is denoted by n(i) i.e. nth element of stack for class i, then the ordered pair [c(i),n(i)] uniquely identifies a node in class stack and correspondingly uniquely identifies an instantaneous face configuration of a Rubik's Hypercube(colored subsquares represent the elements of the feature vector). Thus any hyperedge is a set of these ordered pairs that transcend multiple class stacks.

35.(DONE) The Multiplanar Primordial Field Turing Machine model described by the author in "Theory of Shell Turing Machines" in [http://sites.google.com/site/kuja27/UndecidabilityOfFewNonTrivialQuestions.pdf?attredirects=0] with multiple dimensions seems to be analogous to the above. If dimension info is also added to each plane in above hypergraph then both above formulations and Shell Turing Machines look strikingly similar. One more salient feature of this hypergraph is that each class node stack indirectly encodes timing information for events from bottom to top of the stack and height of each stack independently varies. Thus this is more than just a hypergraph. There also exists a strong similarity with Evocation WordNet (one word "reminding" or "evocative" of the other) - http://wordnet.cs.princeton.edu/downloads.html - difference being above is a hypergraph of facts or concepts rather than just words. The accuracy of sense disambiguation is high or even 100% because context is the hyperedge and also depends on how well the hyperedges are constructed connecting the class stack vertices.

36.(DONE) Instead of a wordnet if a relationship amongst logical statements (First Order Logic etc.,) by logical implication is considered then the above becomes even more important as this creates a giant Concept Wide Web as mentioned in [http://sourceforge.net/projects/acadpdrafts/files/ImplicationGraphsPGoodEquationAndPNotEqualToNPQuestion_excerpts.pdf/download] and thus is an alternative knowledge representation algorithm.

37. Implication graphs of logical statements can be modelled as a Random Growth Network where a graph is randomly grown by adding new edges and vertices over a period of learning.

38. If a statistical constraint on the degree of this graph is needed, power law

distributions (Zipf, Pareto etc.,) can be applied to the degrees of the nodes along with Preferential Attachment of newborn nodes.

39.Set cover or Hitting Set of Hypergraph is a Transversal of hypergraph which is a subset of set of vertices in the hypergraph that have non-empty intersection with every hyperedge. Transversal of the above Concept Wide Hypergraph is in a way essence of the knowledge in the hypergraph as in real world what this does is to grasp some information from each document (which is a hyperedge of class vertices) and produces a "summary nucleus subgraph" of the knowledge of the encompassing hypergraph.

40. Random walk on the above concept hypergraph, Cover time etc., which is a markov process. Probably, Cover time of the hypergraph could be the measure of time needed for learning the concept hypergraph.

41. Tree decomposition or Junction Trees of concept hypergraph with bounded tree width constant.[If Junction tree is constructed from Hypergraph modelled as Bayesian Network, then message passing between the treenodes can be implemented, but still needs to be seen as what this message passing would imply for a junction tree of concept hypergraph above.]

42. 2(or more)-dimensional random walk model for thinking process and reasoning (Soccer model in 2-dimensions):

As an experimental extension of point 40 for human reasoning process, random walk on non-planar Concept hypergraph (collective wisdom gained over in the past) or a planar version of it which is 2-dimensional can be theorized. Conflicts in human mind could mimick the bipartite competing sets that make it a semi-random walk converging in a binary "decision". Semi randomness is because of the two competing sets that drive the "reasoning" in opposite directions. If the planar graph is generalized as a complex plane grid with no direction restrictions (more than 8), and the direction is anything between 0 to 2*pi radians, then the distance d after N steps is sqrt(N) (summation of complex numbers and the norm of the sum) which gives an expression for decision making time

in soccer model.(related:
http://web.archive.org/web/20041210231937/http://oz.ss.uci.edu/237/readings/EBRW_nosofsky_1997.pdf). In other words, if mental conflict is phrased as "sequence of 2 bipartite sets of thoughts related by causation" then "decision" is dependent on which set is overpowering. That is the corresponding graph vertices are 2-colored, one color for each set and yet not exactly a bipartite graph as edges can go between nodes of same set. Brownian Motion is also a related to this.

##
C. Slightly Philosophical - Inferences from conflicting opinions:
##

42. (DONE) As an example, if there are "likes" and "dislikes" on an entity which could be anything under universe - human being, machine, products, movies, food etc., then a fundamental and hardest philosophical question that naturally has evaded an apt algorithm: Is there a way to judge an entity in the presence of conflicting witnesses - "likes" and "dislikes" - and how to ascertain the genuineness of witnesses. In http://arxiv.org/abs/1006.4458 and http://www.nist.gov/tac/publications/2010/participant.papers/CMI_IIT.proceedings.pdf this has been the motivation for Interview Algorithm and P(Good) error probability for majority voting which is the basis for this. Moreover the opinions of living beings are far from correct due to inherent bias and prejudices which a machine cannot have.

43. (DONE) Another philosophical question is what happens to the majority voting when every voter or witness is wrong intentionally or makes an error by innocous mistake. [http://sourceforge.net/projects/acadpdrafts/files/ImplicationGraphsPGoodEquationAndPNotEqualToNPQuestion_excerpts.pdf/download]. This places a theoretical limitation on validity of "perceptional judgement" vis-a-vis the "reality".

44. There is a realword example of the above - An entity having critical acclaim does not have majority acclaim often. Similarly an entity having majority acclaim does not have critical acclaim. Do these coincide and if yes, how and when? This could be a Gaussian where tails are the entities getting imperfect judgements and the middle is where majority and critical acclaim coincide.

45. Items 40,41,42 present the inherent difficulty of perfect judgement or perfect inference. On a related note,Byzantine algorithms have similar issues of deciding on a course of action in the presence of faulty processors or human beings and faulty communications between them. Thus, probably above problem reduces to Byzantine i.e Faulty human beings or machines vote "like" or "dislike" on an entity and a decision has to be taken.

46(THEORY). Would the following experimental gadget work? It might, but impractical:
A voter gadget spawns itself into liker, disliker and neutral parallel threads on an entity to be judged. The neutral thread gathers inputs from liker and disliker threads and gets a weighted sum on them to classify the entity as "like-able" or "dislike-able". Thus real-world classification or judgement and perception seem mythical. (And how is weightage decided? Can this be using SentiWordNet score?)

47(THEORY). Evocation is a realworld everyday phenomenon - a word reminding of the other. Positive thought chooses the right evocation and a negative thought chooses the negative evocation in the absence of context to disambiguate. For example, for a pessimist "fall" means "falling down" while for an optimist "fall" could mean "windfall or sudden gain". For a pestimist((optimist+pessimist)/2), fall could mean one of the "seasons" or even nothing. Mind sees what it chooses to see. Pessimism and Optimism are probably mathematically definable as accumulated weighted thoughts of past occurrences and any future thought or decision is driven by this "accumulated past" which acts as an implicit "disambiguator". This accumulated past or a Karma is reminiscent of above hypergraph of class stacks.

48(THEORY). A crude way to automatically disambiguate is to lookup the corresponding

class stack and analyze only bounded height of events from the top of the stack. Bounded height would imply analyzing only a small window of the past events. For example, looking up the class stack from top to bottom for limited height of 2 for "fall" might have hyperedges "tree fall, heavy rain fall". Thus automatic disambiguation context could be the tuple of tuples [[tree], [heavy, rain]] gathered from each hyperedge analyzed from top. Then this tuple of tuples has to be weighted to get majority. Alternatively, each tuple in this tuple set, as a random variable, could be assigned a belief potential or a conditional probability of occurrence based on occurrence of other tuples in the set. Thus a Bayesian Belief Network can be constructed and by propagating belief potential, probability of each tuple can be computed. Most disambiguating tuple is the one with high belief propagation output. Alternatively, elements of the set of tuples above can be construed as Observations and the State - the most disambiguating tuple - is Hidden and needs to be measured. Thus a Hidden Markov Model can be built. Emission probabilities for an Observation at a State are to be given as priors - probability of a tuple being observed for a word or "class". Importantly each stack is grown over time and is a candidate for Markov process with the restriction - node at height h is dependent only on node at height (h-1) - height encodes timestamp. (But this makes the inference semi-statistical). Consequently, this gives a Trellis from which a Viterbi path can be extracted. Forward-Backward probabilities can be computed (Either generative or discriminative model can be applied, generative being costlier) using this Markov property and deeming the stack itself as a dynamically grown Hidden Markov Model where the Observations are the tuples(hyperedges) and the hidden state is the document that disambiguates. Using the Forward-Backward algorithm, the State(hyperedge) which maximizes the Probability of ending up in that State is the most disambiguating document hyperedge = argmax [Pr(State(t)=Hyperedge(e) / ObservedTuples(1..t)] i.e The tuple or hyperedge that has highest forward conditional probability in the Trellis transition. This assumes the Hypergraph node stack as a Markov Chain. Moreover, State need not be a hyperedge. State is rather a "meaningful context". As in above example, uttering "fall" would kickoff a Forward-Backward computation on the Hypergraph node class stack for "fall" considering the stack as a Hidden Markov Model and would output the argmax State (which is not limited to hyperedge) as above. (Handwritten illustration at:
https://sites.google.com/site/kuja27/NotesOnConceptHypergraphHMM_and_ImplicationGr

aphConvexHulls_2013-12-30.pdf?attredirects=0&d=1)

49(THEORY). Thus Sentiment Mining techniques can be used for the above hypergraph to get the overall sentiment of the hypergraph past. Probably each hyperedge might have to be analyzed for positive or negative sentiment to get grand total. Citation graph maxflow in http://arxiv.org/abs/1006.4458 already gives a SentiWordnet based algorithm to weight each edge of a citation graph. Citations can be generalized as "like" or "dislike" opinions on entities as mentioned above.

50(THEORY). Thus not only documents but also events in human life can be modelled as above hypergraph to get a graphical model of the past and a PsychoProfiling of an individual could be arrived at using Sentiment mining. Thus, item 12 about "AstroPschoAnalysis" is feasible by two parallel paths which converge - Psychoprofiling by above concept or event hypergraph for a human being and equating it with results from Astronomical+Astrological analysis done through String Multiple Sequence Alignment mining.

##
##########
51. Initial Design Notes for - Mundane Predictive Model:
##
##########

- (DONE) DecisionTree, NaiveBayes and SVM classifiers and horoscope encoders are already in the AstroInfer codebase.
- (DONE) Encode the dataset which might be USGS or NOAA(Science on a Sphere) datasets or anyother dataset available after getting text horos for these using Maitreya's Dreams textclient (AstroInfer version)
- (DONE) Above gives encoded horoscope strings for all classes of mundane events in both Zodiacal and AscendantRelative formats (autogenerated by Python scripts in python-src/)
- (DONE) Above set is a mix of encoded strings for all events which can be classified

using one of the classifiers above.
- (DONE) For each class the set of encoded horo strings in that class can be pairwise or multiple-sequence aligned to get the pattern for that class
- (DONE) There are two sets for each class after running the classifiers - one set is AscendantRelative and the other is Zodiacal
- (DONE) The above steps give the mined astronomical patterns for all observed mundane events - Pattern_Mundane_Observed_AscRelative and Pattern_Mundane_Observed_Zodiacal

52. Above has implemented a basic Class and Inference Model that was envisaged 10 years ago in 2003. Class is the set-theoretic notion of sets sharing common underlying theme. Using VC Dimension is a way to determine accuracy of how well the dataset is "shattered" by the classes.

53. Now on to mining the classics for patterns: For all classes of events, running the classifier partitions the rules in a classic into set of rules or patterns for that class. Here again there are two sets for each class - Pattern_Mundane_Classic_AscRelative and Pattern_Mundane_Classic_Zodiacal

54. Thus correlation of the two sets *_Observed_* and *_Classic_* (each set has 2 subsets) for percentage similarity using any known similarity coefficient would unravel any cryptic pattern hidden astronomical datasets for that event and would be a circumstantial and scientific proof of rules in astrological classics with strong theoretical footing and would also bring to light new rules earlier unknown.

55. More importantly, for example, if a non-statistical,astrological model of rainfall is computed based on a classical rule which says that "Venus-Mercury in single sign or Venus-Sun-Mercury in close proximity degrees with Sun in the middle causes copious rainfall" is used as a model, expected output of the model could be "Between 28October2013 and 15December2013 there could be peak monsoon activity in ----- longitude ---- latitude with --- percentage probability". And thus this could be a Medium Range Weather Forecasting tool.

56. (ONGOING) Integrate AstroInfer,USB-md and VIRGO into an approximately intelligent cloud OS platform that not just executes code statically but also dynamically learns from the processes (say through log files, executables, execution times etc., and builds predictive models).
USB-md Design Document: http://sourceforge.net/p/usb-md/code-0/HEAD/tree/USBmd_notes.txt
VIRGO Design Document: http://sourceforge.net/p/virgo-linux/code-0/HEAD/tree/trunk/virgo-docs/VirgoDesign.txt
This AsFer+USBmd+VIRGO+KingCobra would henceforth be known as "Krishna iResearch Intelligent Cloud Platform - iCloud "

57. Encoding a document with above Hypergraph of Class stack vertices and Hyperedges - This hypergraph of concepts can also be used as a steganographic encoding tool for obfuscating a document (somewhat an encryption of different kind). For example, each document is a hyperedge in this hypergraph of class stack vertices. By choosing a suitable encoding function every concept or a word can be replaced with some element within each class stack vertex. This encoding function chooses some element in each class stack - f(word or concept C1 in a document) = a different word or concept C2 in the class stack vertex that contains C1. This function f has to be one-one and onto so that it is invertible to get the original document. This function can be anything modulo the height of that stack. Thus a plain text meaningful document can encrypt another meaningful hidden document without any ciphertext generation.

58. An automaton or Turing machine model for data or voice (or musical) samples - Voice samples or musical notations are discrete in time and occur in stream. As an experimental research, explore on possibility of automatically constructing an automaton or Turing machine that recognizes these languages (RE or CFG).

59. Alternatively, a Discrete Fourier Transform on a set of data or voice samples gives a set of frequencies - a traditional DSP paradigm.

60. Experimental Idea of items 56 and 57 is to mine patterns in discrete data and voice(music for example) samples. Music as an expression of mathematics is widely studied as Musical Set Theory - Transpositions and Inversions (E.g http://en.wikipedia.org/wiki/Music_and_mathematics, http://www.ams.org/samplings/feature-column/fcarc-canons, http://www-personal.umd.umich.edu/~tmfiore/1/musictotal.pdf). Items 56,57 and 58 could help mining deep mathematical patterns in classical and other music and as a measure of similarity and creativity.

EventNet and Theoretical analysis of Logical Time:

----

61. In addition to Concept wide hypergraph above, an interesting research could be to create an EventNet or events connected by causation as edge (there is an edge (x,y) if event x causes event y). Each event node in this EventNet is a set of entities that take part in that event and interactions among them.This causation hypergraph if comprehensively constructed could yield insights into apparently unrelated events.This is hypergraph because an event can cause a finite sequence of events one after the other, thereby including all those event vertices. For simplicity,it can be assumed as just a graph.

62. Each event node in the node which is a set as above, can have multiple outcomes and hence cause multiple effects. Outcome of an interaction amongst the elements of an event node is input to the adjacent event node which is also a set. Thus there could be multiple outgoing outcome edges each with a probability. Hence the EventNet is a random, directed, acyclic graph (acyclic assuming future cannot affect past preventing retrocausality). Thus EventNet is nothing but real history laid out as a graph. Hypothetically, if every event from beginning of the universe is causally connected as above, an indefinite ever growing EventNet is created.

63. If the EventNet is formulated as a Dynamic Graph with cycles instead of Static Graph

as above where there causal edges can be deleted, updated or added, then the above EventNet allows changing the past theoretically though impossible in reality. For example, in an EventNet grown upto present, if a causal edge is added from a present node to past or some causal edge is deleted or updated in the past, then "present" causes "past" with or without cycles in the graph.

64. EventNet can be partitioned into "Past","Present" and "Future" probably by using some MaxFlow-MinCut Algorithms. The Cut of the graph as Flow Network is the "Present" and either side of the Cut is "Past" and "Future".

65. A recreational riddle on the EventNet: Future(Past) = Present. If the Future is modelled as a mathematical function, and if Past is fixed and Present is determined by 100% freewill and can have any value based on whimsical human actions, then is the function Future() well-defined?

66. Conjecture: Undirected version of EventNet is Strongly Connected or there is no event in EventNet without a cause (outgoing edge) or an effect(incoming edge).

67. Events with maximum indegree and outdegree are crucial events that have deep impact in history.

68. Events in each individual entity's existence are subgraphs of universal EventNet. These subgraphs overlap with each other.

69. There is an implicit time ordering in EventNet due to each causation edge. This is a "Logical Clock" similar to Lamport's Clock.

70. EventNet can also be viewed as a Bayesian Network.

----

71. Mining EventNet for Patterns:

As an old saying goes "history repeats itself". Or does it really? Are there cycles of events? Such questions warrant checking the EventNet for cycles. But the above EventNet is acyclic by definition.This probably goes under the item 48 above that models the events as a hypergraph based on classification of events which is different from EventNet altogether. Alternatively, the EventNet nodes which are events with set of partakers and their interactions,can be mined for commonalities amongst them. Thus checking any pair of event nodes separated by a path of few edges (and thus separated in logical time) for common pattern suffices and this recursively continues.

Frequent Subgraph Mining of above EventNet (i.e whether a subgraph "repeats" with in the supergraph) as mentioned in Graph Search feature above could find patterns in events history.(Reference: Graph growing algorithms in chapter "Graph Mining", Data Mining, Han and Kamber)

----
72. Fractal nature of events:

Are events self-similar or exhibit fractal nature? This as in item 71, needs mining the event nodes which are sets of partakers for self-similarity. A fractal event is the one which has a cycle and any arbitrary point on the cycle has a cycle attached at that point and so on. An example of fractal event is the Hindu Vedic Calendar that has Major cycles, subcycles, cycles within cycles that occur recursively implying that calendar is self-similar. How to verify if the EventNet graph has a fractal nature or how to prove that a graph is "Fractal" in general? Would it be Ramsey theory again?

73. EventNet is a Partial Order where there may not be edges of causality between some elements. Thus a Topological Sort on this EventNet directed,acyclic graph gives many possible orderings of events. If history of the universe assuming absolute time is formulated as EventNet, then the topological sort does not give a unique ordering of events which is counterintuitive to absolute time.(Is this sufficient to say there is no absolute time?).

74. EventNet on history of events in the universe is quite similar to Directed Acyclic Graph based Scheduling of tasks with dependencies on parallel computers. Thus universe is akin to a Infinitely Massive Parallel Computer where events are dynamically scheduled with partakers, outcomes and with non-determinism. Even static scheduling is NP-Complete.

75. In the above EventNet, each node which is a set can indeed be a graph also with freewill interactions amongst the set members as edges. This gives a graph G1 with each node being replaced by a graph G2. This is a fractal graph constructed naturally and notions of Outer products or Tensor products or Kronecker products with each entry in adjacency matrix replaced by another adjacency matrix apply. By definition each node can be replaced by different graph.

76. Similar to Implication graphs as Random Growth Graphs, EventNet also qualifies as a Random Growth Network as events happen randomly and new edges are added whenever events happen (with previous Kronecker Tensor model). Rich-Get-Richer paradigm can also hold where nodes with more adjacent nodes are more likely to have future adjacent nodes. (A related notion of Freewill Interactions in http://sites.google.com/site/kuja27/UndecidabilityOfFewNonTrivialQuestions.pdf?attredirects=0 is worth mentioning here).

77. Regularity Lemma can be used as a tool to test Implication and EventNet Random Growth Graphs - specifically seems to have an application in EventNet Kronecker Graph model above where each node is replaced by a graph and thus can be construed as epsilon-partition of the vertices that differ infinitesimally in density. Each element in the partition is the event.

****
*

78. COMMIT RELATED NOTES

****
*

commits as on 3 September 2013

DecisionTree, NaiveBayes and SVM classifier code has been integrated into AstroInfer toplevel invocation with boolean flags

commits as on 6 September 2013

In addition to USGS data, NOAA Hurricane dataset HURDAT2 has been downloaded and added to repository. Asfer Classifier Preprocessor script has been updated to classify set of articles listed in articlesdataset.txt using NaiveBayesian or DecisionTree Classifiers and training and test set text files are also autogenerated. New parser for NOAA HURDAT2 datset has been added to repository. With this datasets HTML files listed in articlesdataset.txt are classified by either classifiers and segregated into "Event Classes".Each dataset document is of proprietary format and requires parser for its own to parse and autogenerate the date-time-longlat text file.Date-time-longlat data for same "Event Class" will be appended and collated into single Date-time-longlat text file for that "Event Class".

commits as on 12 September 2013

1.asfer_dataset_segregator.py and asfer_dataset_segregator.sh - Python and wrapper shell script that partitions the parsed datasets which contain date-time-long-lat data based on classifier output grepped by the invoker shell script - asfer_dataset_segregator.sh - and writes the names of parsed dataset files which are classified into Event Class "<class>" into text files with names of regular expression "EventClassDataSet_<class>.txt"

2.DateTimeLongLat data for all datasets within an event class (text file) need to be collated into a single asfer.enchoros.<classname>.zodiacal or asfer.enchoros.<classname>.ascrelative. For this a Python script MaitreyaToEncHoroClassified.py has been added to repository that reads the segregated parsed dataset files generated by asfer_dataset_aggregator.sh and invokes maitreya_textclient for all date-time-long-lat data within all parsed datasets for a particular event class - "EventClassDataset_<class>.txt" and also creates autogenerated asfer.enchoros.<class>.zodiacal and asfer.enchoros.<class>.ascrelative encoded files

commits as on 2 November 2013

Lot of commits for implementation of Discrete Hyperbolic Factorization with Stirling Formula Upperbound (http://sourceforge.net/p/asfer/code/HEAD/tree/cpp-src/miscellaneous/DiscreteHyperbolicFactorizationUpperbound.cpp) have gone in. Overflow errors prevent testing large numbers.(URLs:

1) Multiple versions of Discrete Hyperbolic Factorization uploaded in http://sites.google.com/site/kuja27/

2) Handwritten by myself - http://sourceforge.net/projects/acadpdrafts/files/DiscreteHyperbolicFactorization_UpperboundDerivedWithStirlingFormula_2013-09-10.pdf/download and

3) Latex version - http://sourceforge.net/projects/acadpdrafts/files/DiscreteHyperbolicPolylogarithmicSieveForIntegerFactorization_updated_rectangular_interpolation_search_and_StirlingFormula_Upperbound.pdf/download
)

commits as on 20 November 2013 and 21 November 2013
----
1. Lots of testing done on Discrete Hyperbolic Factorization sequential implementation and logs have been added to repository.

2. An updated draft of PRAM NC version of Discrete Hyperbolic Factorization has been uploaded at: http://sourceforge.net/projects/acadpdrafts/files/DiscreteHyperbolicPolylogarithmicSieveForIntegerFactorization_PRAM_TileMergeAndSearch_And_Stirling_Upperbound.pdf/download that does PRAM merge of discrete tiles in logarithmic time before binary search on

merged tile.

3. Preliminary Design notes for CRCW PRAM implementation of the above is added to repository at: http://sourceforge.net/p/asfer/code/237/tree/ImplementationDesignNotesForDiscreteHyperbolicFactorizationInPRAM.jpg. This adds a tile_id to each tile so that during binary search, the factors are correctly found since coordinate info gets shuffled after k-tile merge.

79.Tagging as on 16 December 2013

AsFer version 12.0 and VIRGO version 12.0 have been tagged on 6 December 2013 (updated the action items above).

80. More reference URLs for Parallel RAM design of above NC algorithm for Discrete Hyperbolic Factorization for merging sorted lists:

1.http://arxiv.org/pdf/1202.6575.pdf?origin=publication_detail
2.https://electures.informatik.uni-freiburg.de/portal/download/3/6951/thm15%20-%20parallel%20merging.ppt (Ranks of elements)
3.http://cs.brown.edu/courses/csci2560/syllabus.html (Lecture Notes on CREW PRAM and Circuit Equivalence used in the NC algorithm for Discrete
Hyperbolic Factorization above -
http://cs.brown.edu/courses/csci2560/lectures/lect.24.pdf)
4.http://cs.brown.edu/courses/csci2560/lectures/lect.22.ParallelComputationIV.pdf
5.http://www.cs.toronto.edu/~bor/Papers/routing-merging-sorting.pdf
6.http://www.compgeom.com/~piyush/teach/AA09/slides/lecture16.ppt
7.Shift-and-subtract algorithm for approximate square root computation implemented in Linux kernel (http://lxr.free-electrons.com/source/lib/int_sqrt.c) which might be useful in

finding the approximate square root in discretization of hyperbola (Sequential version of Discrete Hyperbolic Factorization)
8.http://research.sun.com/pls/apex/f?p=labs:bio:0:120 - Guy Steele - approximate square root algorithm
9.http://cstheory.stackexchange.com/questions/1558/parallel-algorithms-for-directed-st-connectivity - related theoretical discussion thread on PRAM algorithms for st-connectivity
10.PRAM and NC algorithms - http://www.cs.cmu.edu/afs/cs/academic/class/15750-s11/www/handouts/par-notes.pdf

----

80.Some notes on extensions to Integer partitions and Hash Functions (https://sites.google.com/site/kuja27/IntegerPartitionAndHashFunctions.pdf?attredirects=0)

----

1. Riemann sums (discrete approximation of Riemann integral) of all the functions corresponding to the hash functions are same. Thus all such functions form an equivalence class.(Assuming each partition created by the hash functions as a function plot)

2. Hardy-Ramanujan asymptotic bound for partition function p(n) is ~ O(e^(pi*sqrt(0.66*n))/(4*1.732*n) which places a bound on number of hash functions also.(http://en.wikipedia.org/wiki/Partition_(number_theory))

3. If m-sized subsets of the above O(m!*e^(sqrt(n))/n) number of hash functions are considered as a (k,u)-universal or (k,u)-independent family of functions(Pr(f(x1)=y1...) < u/m^k,then follwing the notation in the link mentioned above, this m-sized subset family of hash functions follow the Pr(f(x1)=y1 & ...) < u/m^n where n is number of keys and m is the number of values. (m! is for summation over (m,lamda(i)) for all partitions)

4. Thus deriving a bound for number of possible hash functions in terms of number of keys and values could have bearing on almost all hashes including MD5 and SHA.

5. Birthday problem and Balls and Bins problem - Since randomly populating m bins with n balls and probability of people in a congregation to have same birthday are a variant of Integer partitioning and thus hash table bucket chaining, bounds for birthday problem and Chernoff bounds derived for balls and bins could be used for Hash tables also (http://en.wikipedia.org/wiki/Birthday_problem, http://www.cs.ubc.ca/~nickhar/W12/Lecture3Notes.pdf)

6. Restricted partitions which is the special case of integer partitions has some problems which are NP-complete. Money changing problem which is finding number of ways of partitioning a given amount of money with fixed denominations(Frobenius number) is NP-complete(http://citeseer.uark.edu:8080/citeseerx/showciting;jsessionid=92CBF53F1D9823C47F64AAC119D30FC4?cid=3509754, Naoki Abe 1987). Number of partitions with distinct and non-repeating parts follow Roger-Ramanujan identities (2 kinds of generating functions).

7. The special of case of majority voting which involves integer partitions described in https://sites.google.com/site/kuja27/IndepthAnalysisOfVariantOfMajorityVotingwithZFAOC_2014.pdf requires a hash function that non-uniformly distributes the keys into hashes so that no two chains are equal in size (to simulate voting patterns without ties between candidates). This is the special case of restricted partitions with distinct and non-repeating parts of which money changing is the special case and finding a single solution is itself NP-complete.

8. Thus Majority voting can be shown to be NP-complete in 2 ways:

8.1 by Democracy circuit (Majority with SAT) in http://sourceforge.net/projects/acadpdrafts/files/ImplicationGraphsPGoodEquationAndPNotEqualToNPQuestion_excerpts.pdf/download and https://sites.google.com/site/kuja27/PhilosophicalAnalysisOfDemocracyCircuitAndPRGChoice_2014-03-26.pdf

8.2 by reduction from an NP-hard instance of Restricted Partition problem like Money changing problem for Majority voting with constituencies described in https://sites.google.com/site/kuja27/IndepthAnalysisOfVariantOfMajorityVotingwithZFAOC

_2014.pdf

9. Infact the above two ways occur in two places in the process of democratic voting: The democracy circuit is needed when a single candidate is elected while the restricted partition in second point is needed in a multi-partisan voting where multiple candidates are voted for.

10. Point 8.2 above requires a restricted partition with distinct non-repeating parts. There are many results on this like Roger-Ramanujan identities, Glaisher theorem and its special case Euler's theorem which equate number of partitions with parts divisible by a constant and distinctiveness of the parts (odd, differing by some constant etc.,). Such a restricted partition is needed for a tiebreaker and hence correspond bijectively to hash collision chaining.

11. An interesting manifestation of point 10 is that nothing in real-life voting precludes a tie and enforces a restricted partition, with no two candidates getting equal votes, where all voters take decisions independent of one another(voter independence is questionable to some extent if swayed by phenomena like "votebank","herd mentality" etc.,) thereby theoretically invalidating the whole electoral process.

12. Counting Number of such restricted partitions is a #P-complete problem - https://www.math.ucdavis.edu/~deloera/TALKS/denumerant.pdf.

13. If a Hash table is recursive i.e the chains themselves are hashtables and so on... then this bijectively corresponds to a recurrence relation for partition function (expressing a partition of a higher integer in terms of lower integer).

14. If the hash table chains are alternatively viewed as Compositions of an integer (ordered partitions) then there are 2^(n-1) maximum possible compositions. (http://en.wikipedia.org/wiki/Composition_(number_theory))

15. In the summation over all parts of partitions derived in

https://sites.google.com/site/kuja27/IntegerPartitionAndHashFunctions.pdf if m==n then it is the composition in point 14 above and thus summation over all parts of partitions is greater than or equal to 2^(n-1) since some permutations might get repeated across partitions. Thus the summation expresses generalized restricted composition(summation_over_all_partitions_of_n((n,lamda(i))>=2^(n-1)).

16. Logarithm of above summation then is equal to (n-1) and thus can be equated to any partition of n. Thus any partition can be written as a series which is the combinatorial function of parts in all individual partitions.

----
81. Updated drafts on Integer partitions and hash function (with points in 80 above) , Circuits for Error probability in Majority Voting
----
1. https://sites.google.com/site/kuja27/IntegerPartitionAndHashFunctions_2014.pdf?attredirects=0&d=1
2. https://sites.google.com/site/kuja27/CircuitForComputingErrorProbabilityOfMajorityVoting_2014.pdf?attredirects=0&d=1

----
82. Reference URLs for restricted integer partitions with distinct parts
----
(for http://sites.google.com/site/kuja27/IntegerPartitionAndHashFunctions_2014.pdf?attredirects=0&d=1)
1. Generating function - http://math.berkeley.edu/~mhaiman/math172-spring10/partitions.pdf
2. Schur's theorem for asymptotic bound for number of denumerants - http://en.wikipedia.org/wiki/Schur's_theorem
3. Frobenius problem - http://www.math.univ-montp2.fr/~ramirez/Tenerif3.pdf
4. Locality Sensitive Hashing (that groups similar keys into a collision bucket chain using standard metrics like Hamming Distance) - http://hal.inria.fr/inria-00567191/en/,

http://web.mit.edu/andoni/www/LSH/index.html, http://en.wikipedia.org/wiki/Locality-sensitive_hashing
5. Locality Sensitive Hashing and Lattice sets (of the diophantine form a1*x1+a2*x2+...+an*xn) - http://hal.inria.fr/docs/00/56/71/91/PDF/paper.pdf
6. Minhash and Jaccard similarity coefficient - http://en.wikipedia.org/wiki/MinHash#Jaccard_similarity_and_minimum_hash_values (similar to LSH that emphasizes on hash collisions for similarity measures between sets e.g bag of words of URLs)

----
83. Reduction from Money Changing Problem or 0-1 Integer LP to Restricted Partitions with distinct parts
----
If the denominations are fixed as 1,2,3,4,5,....,n then the denumerants to be found are from the diaphantine equation:
a1*1 + a2*2 + a3*3 + a4*4 + a5*5 + ...+ an*n
(with ai = 0 or 1). GCD of all ai(s) is 1. Thus Schur's theorem for MCP or Coin Problem applies.
Integet 0-1 LP NP-complete problem can also be reduced to above diophantine format instead of MCP. Finding one such denumerant with
boolean values is then NP-complete and hence finding one partition with distinct non-repeating parts is NP-complete (needed in multipartisan
majority vote).

----
84. Commits as on 23 April 2014

Updated pgood.cpp with some optimizations for factorial computations and batched summation to circumvent overflow to some extent.
For Big decimals IEEE 754 with 112+ bits precision is needed for which boost::multiprecision or java.math.BigDecimal might have to be used.

----

85. Commits as on 7 July 2014

Initial implementation of a Chaos attractor sequence implementation committed to repository.

----

86. Commits as on 8 July 2014

Python-R (rpy2) code for correlation coefficient computation added to above Chaos attractor implementation.

----

87. Commits as on 9 July 2014

DJIA dataset, parser for it and updated ChaosAttractor.py for chaotic and linear correlation coefficient computation of DJIA dataset
have been committed.

----

88. Commits as on 10 July 2014

Python(rpy2) script for computing Discrete Fourier Transform for DJIA dataset has been added (internally uses R). [python-src/DFT.py]

----

89. Commits as on 11 July 2014

Python(rpy2) script for spline interpolation of DJIA dataset and plotting a graph of that using R graphics has been added. [python-src/Spline.py]

Srinivasan Kannan (alias) Ka.Shrinivaasan (alias) Shrinivas Kannan
https://sites.google.com/site/kuja27/

==

```
/*************************************************************************************
USBmd - an experimental USB driver for debugging
```

Personal website(research): https://sites.google.com/site/kuja27/
emails: ka.shrinivaasan@gmail.com, shrinivas.kannan@gmail.com,
kashrinivaasan@live.com

***/

USBmd driver is an experimental modified version of already existing USB driver in linux.

Purpose of this modified version is for doing more sophisticated debugging of USB endpoints and devices and as
USB packet sniffer. Technical Necessity for this was created due to prolonged data theft, id spoofing and cybercrime that has been happening
in author's personal electronic devices for years that resulted in a Cybercrime Police Complaint also few years ago.

There were also such incidents while developing open source code (some code commits have description of these mysterious occurrences). There is no comprehensive USB debugger available on linux to sift bad traffic though there are strong evidences of such cybercrime and datatheft through other sources. Author is inclined to believe that such recurring events of datatheft that defies all logic can have no other intent but to cause malafide theft or loss of private data and an act of defamation among other things.

This is also done as a technical learning exercise to analyze USB Hosts, packets and USB's interaction,if any, with wireless devices including
mobiles, wireless LANs(radiotap) etc.,

In the longterm USBmd might have to be integrated into VIRGO. As VIRGO would would have the synergy of AstroInfer machine learning
codebase for "learning" from datasets, this USBmd driver can have the added ability of analyzing large USB traffic (as a dataset)

using some decision making algorithms and evolve as an anti-cybercrime, anti-plagiarism and anti-theft tool to single out
"malevolent" traffic that would save individuals and organisations from the travails of tampering and loss of sensitive confidential data.

The pattern mining of numeric dataset designed for AstroInfer can apply here also since USB bitstream can be analyzed using algorithms for
numerical dataset mining. Also Discrete Fourier Transform used for analyzing data for frequencies (periodicities if any) can be used for
USB data , for example USB wireless traffic.

==

new UMB driver bind - 27 Feb 2014 (for Bus id 7)

==

Following example commandlines install umb.ko module, unbind the existing option driver from bus-device id and bind the umb.ko to that bus id:

```
sudo insmod umb.ko
echo -n "7-1:1.0" > /sys/bus/usb/drivers/option/unbind
echo -n "7-1:1.0" > /sys/bus/usb/drivers/umb/bind
```

/***

VIRGO - Linux Kernel Extensions For Cloud

Copyright (C):
Srinivasan Kannan (alias) Ka.Shrinivaasan (alias) Shrinivas Kannan
Independent Open Source Developer, Researcher and Consultant
Ph: 9789346927, 9003082186, 9791165980
Open Source Products Profile(Krishna iResearch):
http://sourceforge.net/users/ka_shrinivaasan,
http://www.ohloh.net/accounts/ka_shrinivaasan
Personal website(research): https://sites.google.com/site/kuja27/
emails: ka.shrinivaasan@gmail.com, shrinivas.kannan@gmail.com,
kashrinivaasan@live.com

***/

Goal of Virgo is to design a module extension with cloud capabilities with cpu and memory pooling i.e the user created threads get executed transparently across machines in the cloud(thus this in a way pools the CPU and memory resources on the cloud). Presently there seems to be no implementation with fine-grained support for thread execution on the cloud though there are coarse grained clustering and SunRPC implementations available.

Memory pooling:

Memory pooling is proposed to be implemented by a new virgo_malloc() system call that

transparently allocates a block of virtual memory from memory pooled from virtual memory scattered across individual machines part of the cloud.

CPU pooling or cloud ability in a system call:

Clone() system call is linux specific and internally it invokes sys_clone(). All fork(),vfork() and clone() system calls internally invoke do_fork(). A new system call virgo_clone() is proposed to create a thread transparently on any of the available machines on the cloud.This creates a thread on a free or least-loaded machine on the cloud and returns the results.

virgo_clone() is a wrapper over clone() that looks up a map of machines-to-loadfactor and get the host with least load and invokes clone() on a function on that gets executed on the host. Usual cloud implementations provide userspace API that have something similar to this - call(function,host). Loadfactor can be calculated through any of the prominent loadbalancing algorithm. Any example userspace code that uses clone() can be replaced with virgo_clone() and all such threads will be running in a cloud transparently.Presently Native POSIX threads library(NPTL) and older LinuxThreads thread libraries internally use clone().

Kernel has support for kernel space sockets with kernel_accept(), kernel_bind(), kernel_connect(), kernel_sendmsg() and kernel_recvmsg() that can be used inside a kernel module. Virgo driver implements virgo_clone() system call that does a kernel_connect() to a remote kernel socket already __sock_create()-d, kernel_bind()-ed and kernel_accept()-ed and does kernel_sendmsg() of the function details and kernel_recvmsg() after function has been executed by clone() in remote machine. After kernel_accept() receives a connection it reads the function and parameter details. Using these kthread_create() is executed in the remote machine and results are written back to the originating machine. This is somewhat similar to SunRPC but adapted and made lightweight to suit virgo_clone() implementation without any external data representation.

Experimental Prototype

----
virgo_clone() system call and a kernel module virgocloudexec which implements Sun RPC interface have been implemented.

VIRGO - loadbalancer to get the host:ip of the least loaded node
----
Loadbalancer option 1 - Centralized loadbalancer registry that tracks load:

Virgo_clone() system call needs to lookup a registry or map of host-to-load and get the least loaded host:ip from it. This requires a load monitoring code to run periodically and update the map. If this registry is located on a single machine then simultaneous virgo_clone() calls from many machines on the cloud could choke the registry. Due to this, loadbalancer registry needs to run on a high-end machine. Alternatively,each machine can have its own view of the load and multiple copies of load-to-host registries can be stored in individual machines. Synchronization of the copies becomes a separate task in itself(Cache coherency). Either way gives a tradeoff between accuracy, latency and efficiency.

Many application level userspace load monitoring tools are available but as virgo_clone() is in kernel space, it needs to be investigated if kernel-to-kernel loadmonitoring can be done without userspace data transport.Most Cloud API explicitly invoke a function on a host. If this functionality is needed, virgo_clone() needs to take host:ip address as extra argument,but it reduces transparent execution.

(Design notes for LB option 1 handwritten by myself are at :http://sourceforge.net/p/virgo-linux/code-0/HEAD/tree/trunk/virgo-docs/MiscellaneousOpenSourceDesignAndAcademicResearchNotes.pdf)

Loadbalancer option 2 - Linux Psuedorandom number generator based load balancer(experimental) instead of centralized registry that tracks load:

Each virgo_clone() client has a PRG which is queried (/dev/random or /dev/urandom) to get the id of the host to send the next virgo_clone() function to be executed
Expected number of requests per node is derived as:

expected number of requests per node = summation(each_value_for_the_random_variable_for_number_of_requests * probability_for_each_value) where random variable ranges from 1 to k where N is number of processors and k is the number of requests to be distributed on N nodes

=expected number of requests per node = (math.pow(N, k+2) - k*math.pow(N,2) + k*math.pow(N,1) - 1) / (math.pow(N, k+3) - 2*math.pow(N,k+2) + math.pow(N,k+1))

This loadbalancer is dependent on efficacy of the PRG and since each request is uniformly, identically, independently distributed use of PRG
would distribute requests evenly. This obviates the need for loadtracking and coherency of the load-to-host table.

(Design notes for LB option 2 handwritten by myself at :http://sourceforge.net/p/virgo-linux/code-0/HEAD/tree/trunk/virgo-docs/MiscellaneousOpenSourceDesignAndAcademicResearchNotes.pdf)

(python script in virgo-python-src/)

****

VIRGO ToDo and NiceToHave Features (longterm with no deadline)

****

1. More Sophisticated VIRGO config file and read_virgo_config() has to be invoked on syscall clients virgo_clone and virgo_malloc also. At present config is read by kernel module only which would work only on a single machine.

2. Object Marshalling and Unmarshalling (Serialization) Features

(DONE) 3. Virgo_malloc(), virgo_set(), virgo_get() and virgo_free() syscalls that virtualize the physical memory across all cloud nodes into a single logical memory behemoth (NUMA visavis UMA). (There are random crashes in copy_to_user and copy_from_user in syscall path for VIRGO memory pooling commands that were investigated but turned out to be mystery)
Initial Design Handwritten notes committed at: http://sourceforge.net/p/virgo-linux/code-0/210/tree/trunk/virgo-docs/VIRGO_Memory_Pooling_virgomalloc_initial_design_notes.pdf

4. Integrated testing of Intermodule Kernel Space Execution with sophisticated drivers like video cards, audio cards,etc., and VIRGO cluster testing.

(DONE)5. Multithreading of VIRGO cloudexec kernel module (if not already done by kernel module subsystem internally)

(ONGOING) 6. Sophisticated queuing and persistence of CPU and Memory pooling requests in Kernel Side (by possibly improving already existing kernel workqueues). Either open source implementations like ActiveMQ can be used or Queuing implementation has to be written from scratch or both. ActiveMQ supports REST APIs and is JMS implementation.

(ONGOING) 7. Integration of Asfer(AstroInfer) algorithm codes into VIRGO which would add machine learning capabilities into VIRGO - That is, VIRGO cloud subsystem which is part of a linux kernel installation "learns" and "adapts" to the processes that are executed on VIRGO. This catapults the power of the Kernel and Operating System into an artificially (rather approximately naturally) intelligent computing platform (a software "brain"). For example VIRGO can "learn" about "execution times" of processes and suitably act for future processes. PAC Learning of functions could be theoretical basis for this.

8. A Symmetric Multi Processing subsystem Scheduler that virtualizes all nodes in cloud

(probably this would involve improving the loadbalancer into a scheduler with priority queues)

(ONGOING) 9. Virgo is an effort to virtualize the cloud as a single machine - Here cloud is not limited to servers and desktops but also mobile devices that run linux variants like Android, and other Mobile OSes. In the longterm, Virgo may have to be ported or optimized for handheld devices.

(DONE) 10. Memory Pooling Subsystem Driver - Virgo_malloc(), Virgo_set(), Virgo_get() and Virgo_free() system calls and their Kernel Module Implementations. In addition to syscall path, telnet or userspace socket client interface is also provided for both VIRGO CPU pooling(virgo_clone()) and VIRGO Memory Pooling Drivers.

(DONE) 11. Virgo Cloud File System with virgo_cloud_open(), virgo_cloud_read() , virgo_cloud_write() and virgo_cloud_close() commands invoked through telnet path has been implemented that transcends disk storage in all nodes in the cloud. It is also fanciful feature addition that would make VIRGO a complete all-pervading cloud platform. The remote telnet clients send the file path and the buf to be read or data to be written. The Virgo File System kernel driver service creates a unique Virgo File Descriptor for each struct file* opened by filp_open() and is returned to client. Earlier design option to use a hashmap (linux/hashmap.h) looked less attractive as file desciptor is an obvious unique description for open file and also map becomes unscalable. The kernel upcall path has been implemented (paramIsExecutable=0) and may not be necessary in most cases and all above cloudfs commands work in kernelspace using VFS calls.

(DONE) 12. VIRGO Cloud File System commands through syscall paths - virgo_open(),virgo_close(),virgo_read() and virgo_write(). (But problems similar to copy_from_user and copy_to_user as in memory pooling syscall path mentioned above could resurface). All the syscalls have been implemented with testcases and more bugs fixed. After fullbuild and testing, virgo_open() and virgo_read() work and copy_to_user() is working, but virgo_write() causes kernel panic.

(DONE) 13. VIRGO memory pooling feature is also a distributed key-value store similar to other prominent key-store software like BigTable implementations, Dynamo, memory caching tools etc., but with a difference that VIRGO mempool is implemented as part of Linux Kernel itself thus circumventing userspace latencies. Due to Kernel space VIRGO mempool has an added power to store and retrieve key-value pair in hardware devices directly which otherwise is difficult in userspace implementations.

14. VIRGO memory pooling can be improved with disk persistence for in-memory key-value store using virgo_malloc(),virgo_set(),virgo_get() and virgo_free() calls. Probably this might be just a set of invocations of read and write ops in disk driver or using sysfs.

15. Fault-tolerant features - Program Analysis and Verification features for user code that can find bugs statically.

16. Operating System Logfile analysis using Machine Learning code in AstroInfer for finding patterns of processes execution and learn rules from the log.

17. Implementations of prototypical Software Transactional Memory and LockFree Datastructures for VIRGO memory pooling.

18. Scalability features for Multicore machines - references: (http://halobates.de/lk09-scalability.pdf, http://pdos.csail.mit.edu/papers/linux:osdi10.pdf)

19. Read-Copy-Update algorithm implementation for VIRGO memory pooling that supports multiple simultaneous versions of memory for readers - widely used in redesigned Linux Kernel.

20. Program Comprehension features as an add-on described in : https://sites.google.com/site/kuja27/PhDThesisProposal.pdf?attredirects=0

21. (ONGOING) Implementation of Distributed Systems primitives for VIRGO cloud viz., Logical Clocks, Bakery Algorithm, Termination Detection, Snapshots, Cache Coherency

subsystem etc., Already a simple timestamp generation feature has been implemented for KingCobra requests with <ipaddress>:<localmachinetimestamp> format

22. Non-trivial,quite complex and time consuming, deeper implementation of virgo_malloc() is:
KERNELSPACE: to extend the implementation of SLAB allocator for kmalloc() which creates a kmem_cache(s) of similar sized objects and kmem_cache_alloc() allocates from these caches. Kmem_cache_create() function might have to be extended with a flag for cloud allocation (SLAB_CLOUD_ALLOC) and based on this flag kernel socket invocations to remote machines have to be made to allocate and obtain remote address and return.
USERSPACE: Do the above for sbrk() and brk() implementation.

23.(ONGOING) Cleanup the code and remove unnecessary comments.

24.(ONGOING) Documentation - This design document is also a documentation for commit notes and other build and debugging technical details.

25. (DONE) Telnet path to virgo_cloud_malloc,virgo_cloud_set and virgo_cloud_get has been tested and working. This is similar to memcached but stores key-value in kernelspace (and hence has the ability to write to and retrieve from any device driver memory viz., cards, handheld devices).An optional todo is to write a script or userspace socket client that connects to VIRGO mempool driver for these commands.

26. Augment the Linux kernel workqueue implementation (http://lxr.free-electrons.com/source/kernel/workqueue.c) with disk persistence if feasible and doesn't break other subsystems - this might require additional persistence flags in work_struct and additional #ifdefs in most of the queue functions that write and read from the disk. Related to item 6 above.

27.(DONE) VIRGO queue driver with native userspace queue and kernel workqueue-handler framework that is optionally used for KingCobra and is invoked through VIRGO

cpupooling and memorypooling drivers. (Schematic in
http://sourceforge.net/p/kcobra/code-svn/HEAD/tree/KingCobraDesignNotes.txt and
http://sourceforge.net/p/acadpdrafts/code/ci/master/tree/Krishna_iResearch_opensourcepr
oducts_archdiagram.pdf)

28.(DONE) KERNELSPACE EXECUTION ACROSS CLOUD NODES which
geographically distribute userspace and kernelspace execution creating
a logical abstraction for a cloudwide virtualized kernel:

```
        Remote Cloud Node Client
        (virgo_clone(), virgo_malloc(),virgo_open() telnet and syscalls)
                |
                |
 (Userspace)|
                |------------------------------------Kernel Sockets------------------------------------>
Remote Cloud Node Service
                                                                        (VIRGO cpupooling,
memorypooling, cloudfs, queue, KingCobra drivers)


        |


        |


        |  (Kernelspace execution)


        |


        V
                <------------------------------------Kernel
Sockets--------------------------------------------
                |
                |
                  |
 (Userspace)|
```

29. (DONE) VIRGO platform as on 5 May 2014 implements a minimum set of features and kernelsocket commands required for a cloud OS kernel: CPU virtualization(virgo_clone), Memory virtualization(virgo_malloc,virgo_get,virgo_set,virgo_free) and a distributed cloud file system(virgo_open,virgo_close,virgo_read,virgo_write) on the cloud nodes and thus gives a logical view of one unified, distributed linux kernel across all cloud nodes that splits userspace and kernelspace execution across cloud as above.

****

CODE COMMIT RELATED NOTES

****

VIRGO code commits as on 16/05/2013

----

1. VIRGO cloudexec driver with a listener kernel thread service has been implemented and it listens on port 10000 on system startup
through /etc/modules load-on-bootup facility

2. VIRGO cloudexec virgo_clone() system call has been implemented that would kernel_connect() to the VIRGO cloudexec service listening at
port 10000

3. VIRGO cloudexec driver has been split into virgo.h (VIRGO typedefs), virgocloudexecsvc.h(VIRGO cloudexec service that is invoked by
module_init() of VIRGO cloudexec driver) and virgo_cloudexec.c (with module ops definitions)

4. VIRGO does not implement SUN RPC interface anymore and now has its own virgo ops.

5. Lot of Kbuild related commits with commented lines for future use have been done viz., to integrate VIRGO to Kbuild, KBUILD_EXTRA_SYMBOLS for cross-module symbol reference.

VIRGO code commits as on 20/05/2013
----
1. test_virgo_clone.c testcase for sys_virgo_clone() system call works and connections are established to VIRGO cloudexec kernel module.

2. Makefile for test_virgo_clone.c and updated buildscript.sh for headers_install for custom-built linux.

VIRGO code commits as on 6/6/2013
----
1. Message header related bug fixes

VIRGO code commits as on 25/6/2013

1.telnet to kernel service was tested and found working
2.GFP_KERNEL changed to GFP_ATOMIC in VIRGO cloudexec kernel service

VIRGO code commits as on 1/7/2013
----
1. Instead of printing iovec, printing buffer correctly prints the messages
2. wake_up_process() added and function received from virgo_clone() syscall is executed with kernel_thread and results returned to
virgo_clone() syscall client.

commit as on 03/07/2013

PRG loadbalancer preliminary code implemented. More work to be done

commit as on 10/07/2013

Tested PRG loadbalancer read config code through telnet and virgo_clone. VFS code to read from virgo_cloud.conf commented for testing

commits as on 12/07/2013

PRG loadbalancer prototype has been completed and tested with test_virgo_clone and telnet and symbol export errors and PRG errors have been fixed

commits as on 16/07/2013
----
read_virgo_config() and read_virgo_clone_config()(replica of read_virgo_config()) have been implemented and tested to read the virgo_cloud.conf config parameters(at present the virgo_cloud.conf has comma separated list of ip addresses. Port is hardcoded to 10000 for uniformity across
all nodes). Thus minimal cloud functionality with config file support is in place. Todo things include function pointer lookup in kernel service, more parameters to cloud config file if needed, individual configs for virgo_clone() and virgo kernel service, kernel-to-userspace upcall and execution instead of kernel space, performance tuning etc.,

commits as on 17/07/2013
----
moved read_virgo_config() to VIRGOcloudexec's module_init so that config is read at boot time and exported symbols are set beforehand.
Also commented read_virgo_clone_config() as it is redundant

commits as on 23/07/2013

Lack of reflection kind of facilities requires map of function_names to pointers_to_functions to be executed
on cloud has to be lookedup in the map to get pointer to function. This map is not scalable if number of functions are
in millions and size of the map increases linearly. Also having it in memory is both CPU and memory intensive.
Moreover this map has to be synchronized in all nodes for coherency and consistency which is another intensive task.
Thus name to pointer function table is at present not implemented. Suitable way to call a function by name of the function
is yet to be found out and references in this topic are scarce.

If parameterIsExecutable is set to 1 the data received from virgo_clone() is not a function but name of executable
This executable is then run on usermode using call_usermodehelper() which internally takes care of queueing the workstruct
and executes the binary as child of keventd and reaps silently. Thus workqueue component of kernel is indirectly made use of.
This is sometimes more flexible alternative that executes a binary itself on cloud and
is preferable to clone()ing a function on cloud. Virgo_clone() syscall client or telnet needs to send the message with name of binary.

If parameterIsExecutable is set to 0 then data received from virgo_clone() is name of a function and is executed in else clause
using dlsym() lookup and pthread_create() in user space. This unifies both call_usermodehelper() and creating a userspace thread
with a fixed binary which is same for any function. The dlsym lookup requires mangled function names which need to be sent by
virgo_clone or telnet. This is far more efficient than a function pointer table.

call_usermodehelper() Kernel upcall to usermode to exec a fixed binary that would inturn execute the cloneFunction in userspace

by spawning a pthread. cloneFunction is name of the function and not binary. This clone function will be dlsym()ed
and a pthread will be created by the fixed binary. Name of the fixed binary is hardcoded herein as
"virgo_kernelupcall_plugin". This fixed binary takes clone function as argument. For testing libvirgo.so has been created from
virgo_cloud_test.c and separate build script to build the cloud function binaries has been added.

 - Ka.Shrinivaasan (alias) Shrinivas Kannan (alias) Srinivasan Kannan
   (https://sites.google.com/site/kuja27)

commits as on 24/07/2013
----

test_virgo_clone unit test case updated with mangled function name to be sent to remote cloud node. Tested with test_virgo_clone
end-to-end and all features are working. But sometimes kernel_connect hangs randomly (this was observed only today and looks similar
to blocking vs non-blocking problem. Origin unknown).

- Ka.Shrinivaasan (alias) Shrinivas Kannan (alias) Srinivasan Kannan
  (https://sites.google.com/site/kuja27)

commits as on 29/07/2013

Added kernel mode execution in the clone_func and created a sample kernel_thread for a cloud function. Some File IO logging added to upcall
binaries and parameterIsExecutable has been moved to virgo.h

commits as on 30/07/2013

New usecase virgo_cloud_test_kernelspace.ko kernel module has been added. This exports a function virgo_cloud_test_kernelspace() and is
accessed by virgo_cloudexec kernel service to spawn a kernel thread that is executed in kernel addresspace. This Kernel mode execution
on cloud adds a unique ability to VIRGO cloud platform to seamlessly integrate hardware devices on to cloud and transparently send commands
to them from a remote cloud node through virgo_clone().

Thus above feature adds power to VIRGO cloud to make it act as a single "logical device driver" though devices are in geographically in a remote server.

commits as on 01/08/2013 and 02/08/2013

Added Bash shell commandline with -c option for call_usermodehelper upcall clauses to pass in remote virgo_clone command message as
arguments to it. Also tried output redirection but it works some times that too with a fatal kernel panic.

Ideal solutions are :
1. either to do a copy_from_user() for message buffer from user address space (or)
2. somehow rebuild the kernel with fd_install() pointing stdout to a VFS file* struct. In older kernels like 2.6.x, there is an fd_install code
with in kmod.c (___call_usermodehelper()) which has been redesigned in kernel 3.x versions and fd_install has been removed in kmod.c .
3. Create a Netlink socket listener in userspace and send message up from kernel Netlink socket.

All the above are quite intensive and time consuming to implement.Moreover doing FileIO in usermode helper is strongly discouraged in kernel docs

Since Objective of VIRGO is to virtualize the cloud as single execution "machine", doing

an upcall (which would run with root abilities) is
redundant often and kernel mode execution is sufficient. Kernel mode execution with intermodule function invocation can literally take over
the entire board in remote machine (since it can access PCI bus, RAM and all other device cards)

As a longterm design goal, VIRGO can be implemented as a separate protocol itself and sk_buff packet payload from remote machine
can be parsed by kernel service and kernel_thread can be created for the message.

commits as on 05/08/2013:

Major commits done for kernel upcall usermode output logging with fd_install redirection to a VFS file. With this it has become easy for user space to communicate runtime data to Kernel space. Also a new strip_control_M() function has been added to strip \r\n or " ".

11 August 2013:

Open Source Design and Academic Research Notes uploaded to
http://sourceforge.net/projects/acadpdrafts/files/MiscellaneousOpenSourceDesignAndAcademicResearchNotes_2013-08-11.pdf/download

commits as on 23 August 2013

New Multithreading Feature added for VIRGO Kernel Service - action item 5 in ToDo list above (virgo_cloudexec driver module). All dependent headers changed for kernel threadlocalizing global data.

commits as on 1 September 2013

GNU Copyright license and Product Owner Profile (for identity of license issuer) have

been committed. Also Virgo Memory Pooling - virgo_malloc() related initial design notes (handwritten scanned) have been committed(http://sourceforge.net/p/virgo-linux/code-0/HEAD/tree/trunk/virgo-docs/VIRGO_Memory_Pooling_virgomalloc_initial_design_notes.pdf)

commits as on 14 September 2013

Updated virgo malloc design handwritten nodes on kmalloc() and malloc() usage in kernelspace and userspace execution mode of virgo_cloudexec service (http://sourceforge.net/p/virgo-linux/code-0/HEAD/tree/trunk/virgo-docs/VIRGO_Memory_Pooling_virgomalloc_design_notes_2_14September2013.pdf). As described in handwritten notes, virgo_malloc() and related system calls might be needed when a large scale allocation of kernel memory is needed when in kernel space execution mode and large scale userspace memory when in user modes (function and executable modes). Thus a cloud memory pool both in user and kernel space is possible.

VIRGO virtual addressing

VIRGO virtual address is defined with the following datatype:

```
struct virgo_address
{
        int node_id;
        void* addr;
};
```

VIRGO address translation table is defined with following datatype:

```
struct virgo_addr_transtable
{
        int node_id;
        void* addr;
};
```

VIRGO memory pooling prototypical implementation

VIRGO memory pooling implementation as per the design notes committed as above is to be implemented as a prototype under separate directory
under drivers/virgo/memorypooling and $LINUX_SRC_ROOT/virgo_malloc. But the underlying code is more or less similar to drivers/virgo/cpupooling and $LINUX_SRC_ROOT/virgo_clone.

virgo_malloc() and related syscalls and virgo mempool driver connect to and listen on port different from cpupooling driver. Though all these code can be within cpupooling itself, mempooling is implemented as separate driver and co-exists with cpupooling on bootup (/etc/modules). This enables clear demarcation of functionalities for CPU and Memory virtualization.

Commits as on 17 September 2013

Initial untested prototype code - virgo_malloc and virgo mempool driver - for VIRGO Memory Pooling has been committed - copied and modified from virgo_clone client and kernel driver service.

Commits as on 19 September 2013

3.7.8 Kernel full build done and compilation errors in VIRGO malloc and mempool driver code and more functions code added

Commits as on 23 September 2013

Updated virgo_malloc.c with two functions, int_to_str() and addr_to_str(), using kmalloc() with full kernel re-build.
(Rather a re-re-build because some source file updates in previous build got deleted somehow mysteriously. This could be related to Cybercrime issues mentioned in https://sourceforge.net/p/usb-md/code-0/HEAD/tree/USBmd_notes.txt)

Commits as on 24 September 2013

Updated syscall*.tbl files, staging.sh, Makefiles for virgo_malloc(),virgo_set(),virgo_get() and virgo_free() memory pooling syscalls. New testcase test_virgo_malloc for virgo_malloc(), virgo_set(), virgo_get(), virgo_free() has been added to repository. This testcase might have to be updated if return type and args to virgo_malloc+ syscalls are to be changed.

Commits as on 25 September 2013

All build related errors fixed after kernel rebuild some changes made to function names to reflect their
names specific to memory pooling. Updated /etc/modules also has been committed to repository.

Commits as on 26 September 2013

Circular dependency error in standalone build of cpu pooling and memory pooling drivers fixed and
datatypes and declarations for CPU pooling and Memory Pooling drivers have been segregated into respective header files (virgo.h and
virgo_mempool.h with corresponding service header files) to avoid any dependency error.

Commits as on 27 September 2013

Major commits for Memory Pooling Driver listen port change and parsing VIRGO memory pooling commands have been done.

Commits as on 30 September 2013

New parser functions added for parameter parsing and initial testing on virgo_malloc() works with telnet client with logs in test_logs/

Commits as on 1 October 2013

Removed strcpy in virgo_malloc as ongoing bugfix for buffer truncation in syscall path.

Commits as on 7 October 2013

Fixed the buffer truncation error from virgo_malloc syscall to mempool driver service which was caused by
sizeof() for a char*. BUF_SIZE is now used for size in both syscall client and mempool kernel service.

Commits as on 9 October 2013 and 10 October 2013

Mempool driver kernelspace virgo mempool ops have been rewritten due to lack of facilities to return a
value from kernel thread function. Since mempool service already spawns a kthread, this seems to be sufficient. Also the iov.iov_len in virgo_malloc has been changed from BUF_SIZE to strlen(buf) since BUF_SIZE
causes the kernel socket to block as it waits for more data to be sent.

Commits as on 11 October 2013
----
sscanf format error for virgo_cloud_malloc() return pointer address and sock_release() null pointer exception has been rectified.

Added str_to_addr() utility function.

Commits as on 14 October 2013 and 15 October 2013

Updated todo list.

Rewritten virgo_cloud_malloc() syscall with:

- mutexed virgo_cloud_malloc() loop
- redefined virgo address translation table in virgo_mempool.h
- str_to_addr(): removed (void**) cast due to null sscanf though it should have worked

Commits as on 18 October 2013

Continued debugging of null sscanf - added str_to_addr2() which uses simple_strtoll() kernel function
for scanning pointer as long long from string and casting it to void*. Also more %p qualifiers where
added in str_to_addr() for debugging.

Based on latest test_virgo_malloc run, simple_strtoll() correctly parses the address string into a long long base 16 and then is reinterpret_cast to void*. Logs in test/

Commits as on 21 October 2013

Kern.log for testing after vtranstable addr fix with simple_strtoll() added to repository and still the other %p qualifiers do not work and only simple_strtoll() parses the address correctly.

Commits as on 24 October 2013

Lot of bugfixes made to virgo_malloc.c for scanning address into VIRGO transtable and size computation. Testcase test_virgo_malloc.c has also been modified to do reinterpret cast of long long into (struct virgo_address*) and corresponding test logs have been added to repository under virgo_malloc/test.

Though the above sys_virgo_malloc() works, the return value is a kernel pointer if the virgo_malloc executes in the Kernel mode which is more likely than User mode (call_usermodehelper which is circuitous). Moreover copy_from_user() or copy_to_user() may not be directly useful here as this is an address allocation routine. The long long reinterpret cast obfuscates the virgo_address(User or Kernel) as a large integer which is a unique id for the allocated memory on cloud. Initial testing of sys_virgo_set() causes a Kernel Panic as usual probably due to direct access of struct virgo_address*. Alternatives are to use only long long for allocation unique-id everywhere or do copy_to_user() or copy_from_user() of the address on a user supplied buffer. Also vtranstable can be made into a bucketed hash table that maps each alloc_id to a chained virgo malloc chunks than the present sequential addressing which is more similar to open addressing.

Commits as on 25 October 2013
----
virgo_malloc.c has been rewritten by adding a userspace __user pointer to virgo_get() and virgo_set() syscalls which are internally copied with copy_from_user() and copy_to_user() kernel function to get and set userspace from kernelspace.Header file syscalls.h has been updated with changed syscalls prototypes.Two functions have been added to map a VIRGO address to a unique virgo identifier and viceversa for abstracting hardware addresses from userspace as mentioned in previous commit notes. VIRGO cloud mempool kernelspace driver has been updated to use virgo_mempool_args* instead of void* and VIRGO cloudexec mempool driverhas been updated accordingly during intermodule invocation.The virgo_malloc syscall client has been updated to modified signatures and return types for all mempool alloc,get,set,free syscalls.

Commits as on 29 October 2013

Miscellaneous ongoing bugfixes for virgo_set() syscall error in copy_from_user().

Commits as on 2 November 2013

Due to an issue which corrupts the kernel memory, presently telnet path to VIRGO mempool driver has been
tested after commits on 31 October 2013 and 1 November 2013 and is working but again there is an issue in kstrtoul() that returns the wrong address in virgo_cloud_mempool_kernelspace.ko that gives the address for
data to set.

Commits as on 6 November 2013

New parser function virgo_parse_integer() has been added to virgo_cloud_mempool_kernelspace driver module which is carried over from lib/kstrtox.c and modified locally to add an if clause to discard quotes and unquotes. With this the telnet path commands for virgo_malloc()
and virgo_set() are working. Today's kern.log has been added to repository in test_logs/.

Commits as on 7 November 2013

In addition to virgo_malloc and virgo_set, virgo_get is also working through telnet path after today's commit for "virgodata:" prefix in virgo_cloud_mempool_kernelspace.ko. This prefix is needed to differentiate data and address so that toAddressString() can be invoked to sprintf() the address in virgo_cloudexec_mempool.ko. Also mempool command parser has been updated to strcmp() virgo_cloud_get command also.

Commits as on 11 November 2013

More testing done on telnet path for virgo_malloc, virgo_set and virgo_get commands which work correctly. But there seem to be unrelated
kmem_cache_trace_alloc panics that follow each successful virgo command execution.
kern.log for this has been added to repository.

Commits as on 22 November 2013

More testing done on telnet path for virgo_malloc,virgo_set and virgo_set after commenting kernel socket shutdown code in the VIRGO cloudexec
mempool sendto code. Kernel panics do not occur after commenting kernel socket shutdown.

Commits as on 2 December 2013

Lots of testing were done on telnet path and syscall path connection to VIRGO mempool driver and screenshots for working telnet path (virgo_malloc, virgo_set and virgo_get) have been committed to repository. Intriguingly, the syscall path is suddenly witnessing series of broken pipe erros, blocking errors etc., which are mostly Heisenbugs.

Commits as on 5 December 2013

More testing on system call path done for virgo_malloc(), virgo_set() and virgo_get() system calls with test_virgo_malloc.c. All three syscalls work in syscall path after lot of bugfixes. Kern.log that has logs for allocating memory in remote cloud node with virgo_malloc, sets data "test_virgo_malloc_data" with virgo_set and retrieves data with virgo_get.

VIRGO version 12.0 tagged.

Commits as on 12 March 2014

Initial VIRGO queueing driver implemented that flips between two internal queues: 1) a native queue implemented locally and 2) wrapper around linux kernel's workqueue facility 3) push_request() modified to pass on the request data to the workqueue handler using container_of on a wrapper structure virgo_workqueue_request.

Commits as on 20 March 2014

- VIRGO queue with additional boolean flags for its use as KingCobra queue
- KingCobra kernel space driver that is invoked by the VIRGO workqueue handler

Commits as on 30 March 2014

- VIRGO mempool driver has been augmented with use_as_kingcobra_service flags in CPU pooling and Memory pooling drivers

Commits as on 6 April 2014

- VIRGO mempool driver recvfrom() function's if clause for KingCobra has been updated for REQUEST header formatting mentioned in KingCobra design notes

Commits as on 7 April 2014

- generate_logical_timestamp() function has been implemented in VIRGO mempool driver that generates timestamps based on 3 boolean flags. At present machine_timestamp is generated and prepended to the request to be pushed to VIRGO queue driver and then serviced by KingCobra.

Commits as on 25 April 2014

- client ip address in VIRGO mempool recvfrom KingCobra if clause is converted to host byte order from network byte order with ntohl()

Commits as on 5 May 2014

- Telnet path commands for VIRGO cloud file system - virgo_cloud_open(),

virgo_cloud_read(), virgo_cloud_write(), virgo_cloud_close() has been implemented and test logs have been added to repository (drivers/virgo/cloudfs/ and cloudfs/testlogs) and kernel upcall path for paramIsExecutable=0

Commits as on 7 May 2014

- Bugfixes to tokenization in kernel upcall plugin with strsep() for args passed on to the userspace

Commits as on 8 May 2014

- Bugfixes to virgo_cloud_fs.c for kernel upcall (parameterIsExecutable=0) and with these the kernel to userspace upcall and writing to a file in userspace (virgofstest.txt) works. Logs and screenshots for this are added to repository in test_logs/

Commits as on 6 June 2014

- VIRGO File System Calls Path implementation has been committed. Lots of Linux Full Build compilation errors fixed and new integer parsing functionality added (similar to driver modules). For the timebeing all syscalls invoke loadbalancer. This may be further optimized with a sticky flag to remember the first invocation which might be usually virgo_open syscall to get the VFS descriptor that is used in subsequent syscalls.

Commits as on 3 July 2014

- More testing and bugfixes for VIRGO File System syscalls have been done. virgo_write() causes kernel panic.

7 July 2014 - virgo_write() kernel panic notes:
----
warning within http://lxr.free-electrons.com/source/arch/x86/kernel/smp.c#L121:

```
static void native_smp_send_reschedule(int cpu)
{
        if (unlikely(cpu_is_offline(cpu))) {
                WARN_ON(1);
                return;
        }
        apic->send_IPI_mask(cpumask_of(cpu), RESCHEDULE_VECTOR);
}
```

This is probably a fixed kernel bug in <3.7.8 but recurring in 3.7.8:

- http://lkml.iu.edu/hypermail/linux/kernel/1205.3/00653.html
- http://www.kernelhub.org/?p=3&msg=74473&body_id=72338
- http://lists.openwall.net/linux-kernel/2012/09/07/22
- https://bugzilla.kernel.org/show_bug.cgi?id=54331
- https://bbs.archlinux.org/viewtopic.php?id=156276

- Ka.Shrinivaasan
http://sites.google.com/site/kuja27

```
"VirgoDesign.txt" 213L, 16302C written
```

KingCobra - A Research Software for Distributed Request Service on Cloud with Arbiters

Srinivasan Kannan (alias) Ka.Shrinivaasan (alias) Shrinivas Kannan
Independent Open Source Developer, Researcher and Consultant
Ph: 9789346927, 9003082186, 9791165980
Open Source Products Profile(Krishna iResearch):
http://sourceforge.net/users/ka_shrinivaasan,
https://www.ohloh.net/accounts/ka_shrinivaasan
Personal website(research): https://sites.google.com/site/kuja27/
emails: ka.shrinivaasan@gmail.com, shrinivas.kannan@gmail.com,
kashrinivaasan@live.com

***/

Theoretical Interludes, Design Notes and ToDo (long term with no deadline)

[This is a major research oriented extension of iCloud(AsFer+USBmd+VIRGO) and COBRA project (done by the author in his BE (1995-1999) along with few other esteemed classmates.
http://sourceforge.net/projects/acadpdrafts/files/Excerpts_Of_PSG_BE_FinalProject_COBRA_done_in_1999.pdf/download.)]

1. (THEORY) There is a cloud of nodes which execute a set of services from randomly created clients.

2. (THEORY) This cloud could be on iCloud (AsFer+USBmd+VIRGO) platform or any other opensource cloud platforms like Hadoop Cluster.

3. (THEORY) The Clients are publishers of Service requests which are of many types - miscellaneous types of Service that could be dynamically added through other kernel modules and invoked through a switch-case or embedded in function itself. Identified by unique id(s) for different types of services (for example Problem reports, Suggestions etc.,)

4. (THEORY) The Services on the Cloud are Subscribers to these requests of specific type. Thus this is the conventional publisher-subscriber model.

5. (THEORY) The requests flow through cloud using a workqueue (which could be a lowlevel Linux workqueue or VIRGO queue or some other queuing middleware software like ActiveMQ). The publishers enqueue and Subscribers dequeue the requests.

6. (THEORY) The difference is that the Cloud has nodes that "deceive" or "corrupt".

7. (THEORY) Service requests - are published by the clients in the need of a service which could be defined by markup file. These requests are scheduled and routed by the middleware to competent authority which services it (with or without timeframe) and replies to the client.

8. (THEORY) Problem reports - are published by clients which are "dissatisfied" by the quality of service by the cloud. These are analyzed by "arbiters" in the cloud which find the faulting node(s) and take action. This allows manual intervention but minimizes it.

9. (THEORY) Suggestions - are enhancement requests sent by clients and require manual intervention.

10. (THEORY) Cloud nodes have a Quality of Service metric calculated by a model.

11. (THEORY) The cloud has a reporting structure of nodes - either as a graph or tree. The graph is dynamically reorganized by weighting the Quality of Service of each node.

12. (THEORY) The difficult part of the above is using Arbiters to find "faulty" nodes based on problem reports from clients.

13. (THEORY) Brewer's CAP conjecture proved by [GilbertLynch] as a theorem (still debated) states that only 2 of the 3 (Consistency of data, Avaliability of data and Partition tolerance when some nodes or messages are lost) can be guaranteed and not all 3 are simultaneously achievable.

14. (THEORY) CAP theorem does not seem to apply to the above faulty scenario with corrupt nodes under Consistency or Availablity or Partition Tolerance. This is because a corrupt node can have any 2 of the 3 - it can give consistent data, is available with success response or can make the cloud work with missing data in partition tolerance but yet can "corrupt" the cloud. Probably this needs to be defined as a new attribute called Integrity.

15. (THEORY) As "corruption" is more conspicuous with monetary element, if above services are "charged" with a logical currency (e.g. bitcoin), then corruption in cloud is defineable approximately as (but not limited to)- "Undue favour or harm meted out to a client not commensurate with the charge for the service (or) unreasonable extra logical currency demanded to execute the service of same quality (or) deliberate obstruction of justice to a client with malevolent and unholy collusion with other cloud nodes with feigned CAP".

16. (THEORY) Identifying criminal nodes as in (15) above seems to be beyond the ambit of CAP. Thus CAP with Integrity further places a theoretical limit on "pure" cloud. If Integrity is viewed as a Byzantine problem with faulty or corrupt processes in a

distributed system, and if resilience factor is rf (expected number of faulty nodes), then most algorithms can ensure a "working" cloud only if resilience is ~30% or less (3*rf+1) of the total number of cloud nodes. Probably this could apply to Integrity also that places a limit of 30% on "corrupt nodes" for the Cloud to work with sanity.

17. (THEORY) Analytics on the Problem reports sent to the cloud queue give a pattern of corrupt nodes. Intrinsic Merit ranking with Citation graph maxflow considering cloud as a flow network where a node positively or negatively cites or "opines" about a node, as mentioned in http://arxiv.org/abs/1006.4458 (author's Master's thesis) and http://www.nist.gov/tac/publications/2010/participant.papers/CMI_IIT.proceedings.pdf(published by the author during PhD) give a p2p ranking of cloud nodes that can be used for analysis though may not be reliable fully.

18. (THEORY) Policing the cloud nodes with arbiters - This seems to be limited by CAP theorem and Integrity as above.

19. (THEORY) Brooks-Iyengar algorithm for sensors in all cloud nodes is an improved Byazantine Fault Tolerant algorithm.

20. (THEORY) BitCoin is a Byzantine Fault Tolerant protocol.

21. (THEORY) Byzantine Fault Tolerance in Clouds is described in http://www.computer.org/csdl/proceedings/cloud/2011/4460/00/4460a444-abs.html, http://www.eurecom.fr/~vukolic/ByzantineEmpire.pdf which is more on Cloud of Clouds - Intercloud with cloud nodes that have malicious or corrupt software. Most of the key-value(get/set) implementations do not have byzantine nodes (for example CAP without Byzantine nodes in Amazon Dynamo: http://www.eurecom.fr/~michiard/teaching/slides/clouds/cap-dynamo.pdf)

22. (THEORY) The problem of fact finding or fault finding using a cloud police has the same limitation as the "perfect inference" described in http://sourceforge.net/p/asfer/code/HEAD/tree/AstroInferDesign.txt

23. (THEORY) Reference article on cloud BFT for Byzantine, Corrupt brokers - Byzantine Fault-Tolerant Publish/Subscribe: A Cloud Computing Infrastructure (www.ux.uis.no/~meling/papers/2012-bftps-srdsw.pdf)

----
24. KingCobra messaging request-response design - options

24a. Implementing a message subscription model in kernelspace where clients publish the message that is queued-in to subscribers' queue (Topic like implementation - use of ActiveMQ C implementation if available).

24b. (ONGOING) At present a minimum kernelspace messaging system that queues remote request and handles through workqueue handler is in place. This responds to the client once the Kingcobra servicerequest function finishes processing the request(reply_to_publisher() in KingCobra driver). Unlike the usual messaging server, in which client publishes messages of a particular type that are listened to by interested clients, one option is to continue the status-quo of KingCobra as a peer-to-peer messaging system. Thus every VIRGO node is both a kernelspace messaging client and server that can both publish and listen. Every message in the cloud can have a unique id assigned by a timestamp server (similar to bitcoin protocol) so that each message floating in the cloud is unique across the cloud (or) no two messages on the VIRGO cloud are same. The recipient node executing kingcobra_servicerequest_kernelspace() parses the unique-id (example naive unique-id is <ip-address:port>#localtimestampofmachine) from the incoming remote request and responds to the remote client through kernel socket connection that gets queued-in the remote client and handled similar to incoming remote request. To differentiate request and response-for-request response messages are padded with a string "REPLY:<unique-id-of-message>" and requests are padded with "REQUEST:<unique-id-of-message>". This is more or less similar to TCP flow-control with SEQ numbers but state-less like UDP. Simple analogy is post-office protocol with reference numbers for each mail and its reply. Thus there are chronologically two queues: (1) queue at the remote VIRGO cloud service node for request (2) queue at the remote client for response to the request sent in (1). Thus any cloudnode can have two types of messages - REQUEST and REPLY. Following schematic diagram has been implemented so far.

24c.(DONE) KingCobra - VIRGO queue - VIRGO cpupooling and mempooling drivers interaction schematic diagram:

```
        KingCobraClient ==========>=<REQUEST:id>====================> VIRGO
cpupooling service =====> VIRGO Queue ============> KingCobraService
              ||
                ||
               ||
                 ||
                <================= VIRGO Queue <====== VIRGO cpupooling service
====<REPLY:id>==================================== V

        KingCobraClient ==========>=<REQUEST:id>====================> VIRGO
mempooling service =====> VIRGO Queue ============> KingCobraService
              ||
                ||
               ||
                 ||
                <================= VIRGO Queue <====== VIRGO mempooling
service ====<REPLY:id>==================================== V
```

24d. (ONGOING) kingcobra_servicerequest_kernelspace() distinguishes the "REQUEST" and "REPLY" and optionally persists them to corresponding on-disk filesystem. Thus a disk persistence for the queued messages can either be implemented in 1) VIRGO queue driver 2) workqueue.c (kernel itself needs a rewrite (or) 3) KingCobra driver. Option (2) is difficult in the sense that it could impact the kernel as-a-whole whereas 1) and 3) are modularized. At present Option 3 persistence within KingCobra driver has been implemented.

24e.Above option 24b implements a simple p2p queue messaging in kernel. To get a Topic-like behaviour in VIRGO queue might be difficult as queue_work() kernel function has to be repeatedly invoked for the same work_struct on multiple queues which are subscribers of that message in AMQ protocol. Moreover creating a queue at runtime on need basis looks difficult in kernel which is usually done through some CLI or GUI interface in ActiveMQ and other messaging servers.

25. For the timestamp service, EventNet described in http://sourceforge.net/p/asfer/code/HEAD/tree/AstroInferDesign.txt is a likely implementation choice.

26. (THEORY) MESSAGE-AS-CURRENCY PROTOCOL: If each message payload is also construed as a currency carrier, each message id can be mapped to a unique currency or a coin with fixed denomination. This is similar to each currency note having a serial number as unique id. Uniqueness is guaranteed since there can be only one message (or coin) with that id on the cloud. This simulates a scenaio - "Sender of the message pays the Receiver with a coin having the unique id and Receiver acknowledges receipt". This is an alternative to BitCoin protocol. Double spending is also prohibited since at any point in time the message or "coin" with unique id can be sent by only one node in the cloud. Unique Cloudwide Timestamp server mimicks the functionality of "Mint". There is a difference here between conventional send-receive of messages - Once a message is sent to remote cloud node, no copy of it should exist anywhere in the cloud. That is, every MAC currency message is a cloudwide singleton. In pseudocode this is expressible as:

```
    m1=MAC_alloc(denomination)
    m2=m1 (---- this is disallowed)
```

Linux kernel allocation functions - kmalloc() - have a krefs functionality for reference counting within kernel. Refcount for MAC message can never exceed 1 across cloud for above singleton functionality - this has to be a clause everywhere for any unique MAC id. This requires a cloudwide krefs rather. Buyer decrements cloudwide kref and Seller increments it.

27. (THEORY) SIMULATING A VIRTUAL ECONOMY with above MAC protocol (Message-as-currency): If each message sent is considered as "money" element and cloud nodes and clients are the consumers and producers of "electronic money", the timestamp "Mint" becomes a virtual Federal Reserve or Central Bank that controls the "electronic money" circulation in the cloud. Infact any REPLY messages could be mapped to a Service a client derives by s(p)ending the REQUEST "money message". Thus value(REQUEST) should equal value(REPLY) where value() is a function that measures the value of a money denomination and the value of goods and/or services for that money. For example Rs.10000 or $10000 has no meaning if it doesn't translate into a value (analogy: erstwhile Gold Standard).When the value() function gets skewed phenomena like Inflation arise. Thus above model could also have a notion of value() and "electronic money inflation". Thus any "message money" with a unique id assigned by the cloud unique id(or logical timestamp) server can exist at most in only one node in the cloud. Money trail can be implemented by prefixing a header to the incoming message money in each cloud node that receives the money which traces the "path" taken. Cloud has to implement some Byzantine Fault Tolerant protocol. The value() function to some extent can measure the "deceit" as above. When a Seller replies with Goods or Services of less value() compared to value(REQUEST MAC) then that is starting point of "cloud corruption".

28. (THEORY) TRADING WITH ABOVE KINGCOBRA MAC protocol - somewhat oversimplified:

```
--------------------
|Unique MAC id MINT|
--------------------
         ||
        -----money trail-----------------
        |
        V
        ....
        Buyer ======= sends MAC message (REQUEST id) =======> Seller (stores the
MAC in local cash reserve and prepends money trail)
        ||                                                    ||
        <============ sends the goods and services (REPLY id) ===
```

In the above schematic, money with unique id in cloud reaches a buyer after many buyer-seller transitions called "money trail". The MAC currency is prefixed by each node to create a chain. Buyer then sends a request to the seller through MAC virtual currency and seller replies with goods and services. Seller prepends the money trail chain. When a transaction occurs the whole cloud need not be notified about it except only buyer and seller. MAC Mint could create a bulk of money denominations and circulate them in cloud economy.

29. (THEORY) VALUE FOR ELECTRONIC MONEY: How is above MAC money earned - This again requires linking value to money (as money is not a value by itself and only a pointer to valuable item or resource). Thus any buyer can "earn" MAC money by something similar to a barter.

30. (THEORY) FIXING VALUE FOR MAC MONEY: To delineate corruption as discussed in 27 above with value() disparity between MAC money and REPLY goods and services, an arbiter node in cloud has to "judge" the value of MAC sent by buyer and goods and services from seller and proclaim if corruption exists. Thus value() function itself has to be some kind of machine learning algorithm. This is related to or same as points 12 to 23 above. For example, while buying an item for few million bucks, value() has to take as input the description of the item and calculate the value "ideally" which is difficult. Because there are no perfect references to evaluate and only a weighted average of available market price range has to be taken as a reference which is error-prone. value() function can be recursively defined as("Reductionism"):

|value(i) = summation(value(ingredients of i)) + cost(integrating the ingredients to create item i) |

Obviously the above recursion combinatorially explodes into exponential number of nodes in the recursion tree. Ideally recursion has to go deep upto quarks and leptons that makeup the standard model. If for practical purposes, recursion depth is restricted to t then size of value() tree is O(m^t) where m is average number of ingredients per component. Hence any algorithm computing the value() recursion has to be exponential in time. Computation of value() in the leaf nodes of the recursion is most crucial as they percolate bottom-up. If leaf nodes of all possible items are same (like quarks and leptons making up universe) then such atomic ingredient has to have "same" value for all items. Only the integration cost varies in the levels of the tree.For infinite case, value() function is conjectured to be undecidable - probably invoking some halting problem reduction

31. (THEORY) Buyer-Seller and MAC electronic money transaction schematic:

```
        Buyer          A-------<id><refcnt:0>---------------------> Seller <id><refcnt:1>
(increments refcnt)
        (<id><refcnt:1>  |
         <id><refcnt:0>  |
        after decrement  |
        refcnt           |
        )--------------->
```

Above has to be transactional (i.e atomic across cloud nodes)

32. (THEORY) MAC protocol reaper

Reaper thread in each cloud node harvests the zero refcounted allocations and invokes destructors on them. Same MAC id cannot
have kref count of 1 or above in more than one cloud node due to the transaction

mentioned previously.

Commits as on 1 March 2014

Example java Publisher and Listeners that use ActiveMQ as the messaging middleware have been committed to repository for an ActiveMQ queue instance created for KingCobra. For multiple clients this might have to be a Topic rather than Queue instance. Request types above and a workflow framework can be added on this. This will be a JMS compliant implementation which might slow down compared to a linux workqueue or queue implementation being done in VIRGO.

Commits as on 17 March 2014

KingCobra userspace library and kernelspace driver module have been implemented that are invoked 1) either in usermode by call_usermodehelper()
2) or through intermodule invocation through exported symbols in KingCobra kernel module, by the workqueue handler in VIRGO workqueue implementation.

Commits as on 22 March 2014

Minimalistic Kernelspace messaging server framework with kernel workqueue,handler and remote cloud client has been completed - For this VIRGO clone cpupooling driver has been added a clause based on a boolean flag, to direct incoming request from remote client to VIRGO linux workqueue which is popped by workqueue handler that invokes a servicerequest function on the KingCobra kernel module. (Build notes: To remove any build or symbol errors, Module.symvers from VIRGO queue has to be copied to VIRGO cloudexec and built to get a unified VIRGO cloudexec Module.symvers that has exported symbol definitions for push_request()). End-to-end test with telnet path client sending a request to VIRGO cloudexec service, that gets queued in kernel workqueue, handled by workqueue handler that finally invokes KingCobra service

request function has been done and the kern.log has been added to repository at drivers/virgo/queuing/test_logs/

Commits as on 29 March 2014

Initial commits for KingCobra Request Response done by adding 2 new functions parse_ip_address() and reply_to_publisher() in kingcobra_servicerequest_kernelspace()

Commits as on 30 March 2014

Both VIRGO cpupooling and mempooling drivers have been modified with use_as_kingcobra_service boolean flag for sending incoming remote cloud node requests to VIRGO queue which is serviced by workqueue handler and KingCobra service as per the above schematic diagram and replied to.

Commits as on 6 April 2014

Fixes for REQUEST and REPLY headers for KingCobra has been made in virgo_cloudexec_mempool recvfrom() if clause and in request parser in KingCobra with strsep(). This has been implemented only in VIRGO mempool codepath and not in VIRGO clone.

Commits as on 7 April 2014

New function parse_timestamp() has been added to retrieve the timestamp set by the VIRGO mempool driver before pushing the request to VIRGO queue driver

----

Commits as on 29 April 2014

Intial commits for disk persistence of KingCobra request-reply queue messages have been done with addition of new boolean flag kingcobra_disk_persistence. VFS calls are used to open and write to the queue.