hello and welcome, id like to first off thank all the committee members for making it out this morning.
My name is lee carraher, i am a phd student working under professor phil wilsey working on scalable and distirbuted data analysis algorithms, and specifically data clustering.
the title of this dissetation defense is approximate clustering algorithms for high dimensional distributed and streaming data.
I'd like to thank the university of cincinnati, prof wilsey, and my fellow student researchers for their involvement in this research.
This research was funded in part by NSF grant ACI–1440420.
My wife said not to tell a joke so i wont, but suffice to say i have some pretty awesome technology jokes, so y ou can just ask me after the presentation.

Here is the agenda for today, I will start discussing the broad problem of classification, specifically statistical classification in machine learning, next I will discuss data analysis in general, as a problem and how clustering is related.
following I will present the research goals of this project in particulas. I will then give a brief preview or the algorithm motivations, and general concepts. followed by a brief survey of related work.
I follow this by a background work discussion neccessary to lay the foundation of the RPHash algorithm description for the static data and streaming cases.
I give a first pass of experiments following these two algorithm formulations and which I follow by an alternative algorithm that is along the lines of the rphash formulation but improves its overall stability.
We will then resume our experimentation and results section.
Following we give a brief conclusion and discussion of future directions.
Afterward i would like to open the floor to any questions and provide further explanations if needed.

Classification is a natural question. it is likely an evolved characteristic that give rise to more complex multicellular animals like vertabrates and eventually humans.
we can think of the basic survival questions of classification, we classify whether things are edible or dangerous, or suitable mates.
since we cannot directly store an instance of every situation, we have to build models. these mental models are built around features, so things like leaf shape and teeth size are good attributes to discern certain classifications.
humans are pretty good classifiers for some things, but not all things.

we are quite good at recognizing other humans

but only for a small subset of people, once the grouping grows beyond a few hundred or thousand we cannot keep track of all the connections.
this is mainly a capacity problem.

we are also not very good at classifying things that we dont have an intuitive understanding of, without significant training and further extraction.
this is a genomic of the Staphylococcus bacteria.

So to recap, we are very good at classify inutive things, and can far exceed computer capabilities in our abstraction abilities, however we stumble on things that are less intuitive, or have massive amounts of data.

Here is a brief example of our specific motivating problem.
In short we want to be able to geographically quantify some hypothesis involving a medical condition
but are bound by data security issues among the data sources this data is potentially large, has high dimensionality, is distributed, and has privacy issues that limit our access.

so for the first parts we can turn to classic data analysis. data analysis is useful because it can reveal unseen patterns that may be hidden from human observation due to the aforementioned limitiations.

data clustering is a solution to some of these problems.
clustering build inherent models of attributes simply by associating observations with similar observations, based on some co-occurance of attributes.
classically k-means offers a solution to this problem based on some similarity metric between obsevations.

here is an example 4 means problem, where we clustered 4 clusters of obsevations.
there are dense regions in the centers, with high variance regions on the borders resulting in likely some mis-classification at the boundaries.

The most common optimization metric for kmeans is to optimixe the subset memberships x in C, such that the set observation distances to the set means are minimized.
kmeans itself is in the problem class max-snp, being directly relatable to the graph problem max clique, however the problem is also in APX approximable algorithms in NP-hard problems.
Fortunetly APX problems, have polynomial time algorithm solutions given some acceptable approximation error.

PTAS are great for some things, but not if we consider the exponential growth in dataset sizes.
this is just a small sample from the kdnuggest data science challenges website involving their challenge sets.
worse yet is that some data sets are unbounded such as in the streaming case.

so of course PTAS wont scale for big data.
furthermore the streaming problem means we cannot store all seen data, and have to find a less than linear complexity algorithm for memory.
And, as we previously discussed some problems have special privacy requirements making distributed solutions all the more difficult.

Now we restate our problem goals from the defense proposal
we want an algorithm that is secure and A distributable for static and streaming data
we want a scalable algorithm that must have sub-quadratic growth complexity
and we want to evaluate this algorithm to show that it is comparable to other clustering methods.

our first motivation for rphash comes from work done previously at the univ. cincinnati on parallel nearest neighbor search.
here we give a brief definition of the problem. given a set of vectors, find the k nearest vectors to the given query vector based on some distance funciton.
the cr-NN formulation is an approximate solution where r is the radius of the neighbors and c the probability of missing a neighbor in that bounding radius.

in this slide we have the general LSH NN algorithm.
The goal is to use so called locality senitive hash functions to convert a dataset of vectors into a hash table, that is queriable by a hash that points back to the original vectors.
the query method uses this same has function, to generate a hash to use for finding the stored list of vectors. LSH is central to our clustering algorithm and will be discussed later in much further detail.

the lsh nn algorithm is theoretically optimal for high dimension NN given a good LSH function.
In essence we can see that the LSH function narrows the search space to those vectors that share the same hash.
this reduction is immensely helpful in lowering the computational costs, but degrades when a hash contains a very large number of clusters.
However another way to look at these dense hashes, is to view them as cluster candidates.

this brings us to our hypothesis. we believe that this degenerate case of LSH is useful for clustering we plan to combine LSH with random projection to yield a dense region indentifier , where the high dimensional space is spanned by the observation attributes. we also believe that by relaxing the absolute optimization to a local minima is in practice not much better than finding an approximation of a datasets density modes

we will now briefly discuss some related work.
classically there are lloyd type clustering algorithms, like the cannonical k-means implementation of hartigan and wong,
mean shift that iteratively optimizes a distance kernel, the agglomerative algorithms that greedily merge clusters until some desired stopping criteria
there is also spectral clustering where the spectra of the graph lapplacian are substituted for the observations,
many of these have rather unscalable complexity with the exception of kmeans, so we will focus on that.
later we will disucss tree based clusterin in which the clustering is model by a decision tree in which ground truths are based on the resulting cluster entropies of a decision in the tree.

These algorithms are slightly more atuned to our setting, in that they are more scalable and approximate.
proclus is a projective clustering method as is cluster ensembles, while dbscan clique and clarans are density seeking algorithms

we also review and compare some more recent algorithms for clustering datastreams they are csketch,
which is very similar to our streaming rphash implementation, streaming kmeans, damped sliding window dstream, and biased resavoir sampling.

now we get to our proposed algorithm description.
we present 2 algorithms that follow a common format for static data and streaming data storage architectures.
we will then present some preliminary results for these methods, followed by an observation and extension that improves the stability of both called tree walk RPHash

RPHash is composed of 4 main components.
first is random projection followed by the previously mentioned LSH functions.
we then discuss approximate and exact counting methods. we follow this by the general offline clustering step.

our first component random projection, is a dimensional reduction technique, that for high dimensional data is comparable to the more rigorous principal component reduction.
there are tight bounds for reduction dimensionalities, give by the JL lemma, and we can further reduce these for the purposes of clustering.
We use the DB friendly construction of the RP matrix, however others exist.

Now we discuss the aforementioned LSH functions.
in this slide we present the general LSH definition. briefly an LSH function is a hash that preserves locality, instead of the typical hash that attempts to preserve uniform distribution over the data hash universe.
Options exist to extend the selectivity and inprove the miss rate error.

in this slide we give some real world lsh functions.
the first set of formulations are derived from lattices and error correcting codes.
however there are more directed lsh functions that are defined for specific purposes, and data settings, such as spherical for data on the sphere, p-stable when the distribution is known, and kmeans when the data is clusterable

In this slide we give an intuition of LSH based clustering.
the hyperplanes depicted by dashed lines define some arbitrary partitioning.
from this we get 4 hash functions and 8 partitions.
if we consider only the highest collision probability hash buckets with the most support, we see that green blue red is represented 2 x is likely a good candidate cluster.
we will use this intuition in the following algorithm definition

we have data prerequisitesm k the number of desired clusters, x the data set in m dimensional space, H our defined hash function, P the projection matrix, C the set of hash-¿ counts map, and M the set of centroid vectors.
we also have an offline clusterer that resolves our candidate clusters to our parametric k clusters

the algorithm proceeds by projecting each vector in X using the matrix P.
the projection is hashed by the LSH function H, and a count is accumulated for each hash in the map C.
once all vectors are added to the map C, we sort the map by counts and get the top highest support locality sensitive hashes.

In the second phase we proceed similarly.
however instead of accumulating the hash counts for all vectors, we only consider vectors that are in a high support hash.
Vectors that are in high support hashes are accumulated in the centroid that corresponds to that hash.
following the processing of all vectors, we have an oversample of centroids that correspond to dense regions.
to get k we simply apply a standard clusterer to the over sampled set of density modes.
this concludes the basic RPHash formulation

For streaming rphash we first review the streaming data setting.
in this setting streaming algorithms maintain som sketch of the data be received over time t.
periodically we appline some offline step to the sketch to get our desired output.

a common streaming algorithm that has wide application is the countmin sketch.
this structure maintains a count of an size stream in a bounded data structure, that uses the probability of universal hash collisions in a markov chain to amplify the incorrect counting error bounds.

A diagram of the streaming RPHash algorithm is given in this slide.
as with the standard RPHash algorithm we project and hash the incoming vector.
the count is maintained in a mincount sketch, and a priority queue maintains the top density modes centroids.
Only centroids are accumulated for the top most common lsh hashes.
This results in the potential to lose high support vectors, but in general the centroid will be re discovered in later rounds.

next we give the formal definition.
similar to before are k, x, H, P and M.
we add to it our counting structure C that is accessed by a bounded priority queue to identify top vectors.

in the general formulation vectors x are received in sequence.
we project and hash them, as in standard RPHash.
however we also add the centroid to the priority queue that is sorted by the support of the hash.
immediately following, if the queue is over it's bound, we remove the lowest support vector.
in this way we only store the centroids with highest support.

we now begin our first set of experiments.
First we give preliminaries of defining the datasets, our evaluation metrics, and an optimization search to find the best rphash configurations.
we give results for RPHash and streaming RPHash.

given that clustering is an ill posed problem, we must rely on an ensemble of metrics as opposed to one single metric to compare against other algorithms.
for this we choose the internal metric WCSSE which is the kmeans optimization metric.
we also chose ARI and cluster purity as external metrics to confirm results with a known ground truth.
In addition to cluster quality metrics, we also evaluate memory consumption and runtime, based on wall time.

Our test datasets consist of varous synthetic dataset, generated from gaussian distributtions.
We list some of the features of the synthetic generators as well.
next we have a table with the datasets pertinant attributes.

we now move on to our optimal cnfiguration search. in rphash we have the various LSH algorithms,
number of projected dimensions,
number of projections,
offline clustering
algorithm and degree of blurring, which is a gibbs sampling technique around the projected cluster.
in total there are 6400 different configurations.
we tested them all, agains the listed datasets.
We give the top configuration results for the datasets and RPHash algorithms vs the k-means algorithm.
The plot shows the variable importance for each of the attributes.
We used projection to latent space/partial least squares analysis to establish variable importance as a set of loading vectors.
from the plot we see that offline clcustering had the best payoff for runtime, while the decode and projcetion dimension is of less importance.
blurring is beneficial as well, and number of projections is beneficial but at the cost of runtime.

Now that we have an optimal set of configurations we can begin our experiments agains other algorithms.
we compare against k-Means, Hierarchical Agglomerative clustering with Single, Complete, and Average link as well as Ward's method.
we also compare against Self-organizing Tree Algorithm, k-Means++.
we compare against the Streaming Clustering algorithms: Streaming k-Means, Damped Sliding Window, DStream, Biased Reservoir Sampling

here we give the results of clustering with the standard RPHash.

now we move to experiments with streamingRPHash. for streaming, because we require a large number of vectors, we give results for the indoor localization dataset.
other dataset comparisons are available in the dissertation, but indoor localization is indicative of the other results.
in the plots we see that RPHash performas best for ARI and Purity and WCSSE on the datasets.

if we look at runtime and memory we see that RPHash and streaming kmeans are optimal in memory usage and runtime.

next we see the performance scalability of RPHash for synthetic datasets. RPHash performs mostly optimal in all dimensions with an acception at 633 dimensions.

this precipetates one of our principal results which is further explored in this plot showing the ari of various clustering methods in different dimensions on synthetic data.
here we see the instability of the RPHash and kmeans algorithms. Fixing stability was a preocupationof our experimentation and development for a considerable time, and suggested we needed optimization to fix the run to run variation.
we tested against different projections and counting algorithms but found no variation to stability, suggesting the cause must lie somewhere in the LSH function.

Often a regular partitioning fits the designed data space well.
however, sometimes the partition is either too sparse or too dense, resulting in hash buckets with too high of support, or buckets with low support.
in order to calibrate the partitioning we scaled the lsh funciton by the variance.
there is some benefit in doing this, however it did not fix our stability issues.
for this reason we introduce what we call composable hash functions.
composable hash functions formally stated here, a hash functions that can be combined and desimated into smaller or larger partitioning regions, by adding or subtracting a suffix of the parent hash.
we can then use these nested hashes to establish more precise keys for dense regions and less dense keys for sparse regions.

in this slide we give a basic diagram of what the adaptive hash would do for a single cluster. notice the first random partition depicted in brown, does not effectively partition the region.
we then follow the side of greatest support, and generate another random partion.
notice that the new region contains the first hash, and either a 0 or 1 denoting the second level of the hash.
this process continues until we get the dense region defined by 100111, bound by the red, blue, green and purple regions.

we give this process formall here and suggest a simpler metric for hashing than random hyperplane partitionion, in which we simply use the origin, to define the hyperplane for subsequent dimensions. in the algorithm we see that we favor the denser regions, continuing to bisect the hash until the region only contains one vector, or some predefined stoping depth.

although in principal this seems like a good method, however testing with synthetic data, suggests that it in fact is not.
here we compare the WCSSE of different LSH algorithms. adaptive lsh does poorly throughout, but is slightly more stable.

following this concept and our desire for stability we consider the composability property further.
not only in for a composable function do we know the parent hash we also know the sibling.
so what if we use this relationship, between all hashes.
eventually we get something that looks like a tree, in which arcs represent hyper planes in space,and nodes represent the support of the partition inscribed by the hyper plane.

here is an example to make it a bit clearer.
1, we start with 3 clusters and some noise data. in all there are 26 vectors. we draw the start of our tree, which has no paritions.
2. we draw a random partition. note note a very good one, but thats randomness for you. this gives us two regions one with 21 vectos and another with 5.
3. we draw generate another random hyperplane, this hyperplane does not split anything below D1, but partitions the region above D1 into two regions with support 5 and 16.
4.we continue this process again
5. and again. now we have a pretty well partitioned data space. we could partition further of course.

the tree walk algorithm in its first phase simply generates the tree. but it is large, and if we want to store centroids for each we have to store 2^n centroids! but really we just need to search this tree, we dont need the whole thing, and we dont need it to be exact either. so again we employ the count-min sketch

this leads us to what we call the count min cut tree.
it's simply an augmentation of the partition tree where each node corresponds to a value in the count min sketch.
and because the tree is generative, based on a hash, we actually never need to store the tree, we can simply traverse it and query the sketch as needed

here is formally the tree generation step.
we project as before, but instead we use a composable hash like the sign based hash. at each depth of the hash, for each vector we add to our count min sketch.
this sketch will be full of collisions but we show a method for resolving errors in the dissertation.

our tree traversal follows the method used in CLTree.
from the related research section. however we make one observation that removes a somewhat computationally intense portion of CLTree to avoid splitting clusters, as we saw in our example.
our observation and proof in the dissertation claims that as dimensionality grows the probability of intersecting a spherical cluster diminishes exponentially.
so in our projected 16 or so dimension the probability is miniscule, and doesnt require the cluster cutting check.

we now state the tree traversal algorithm in this case we present C.ids as a complete set, however it can also simply be the enumeration of binary keys up to two to the tree depth.
we traverse the tree, following the denser node, and subtracting it's support from its parent node, until we are left with only k leaf nodes.
The remaining leaf nodes, are then regarded as the densest clusters.
if we use the count sketch to also store the centroids, we can then in one step get both the densest nodes, and the actual centroids.

we continue our experimentation with this new TWRP clustering algorithm and compare it to other methods.

first we look at TWRP and the kmeans++ algorithm on a very large word2vec set from the googlenews dataset, with 3 million topics represented by 300 dimensional dense vectors.
because there is no ground truth, we regard kmeans++ as the ground truth and evaluate our correlation to kmeans++ as we increase the number of vectors. the trend is roughly thatTWRP achieves 95 percent correspondence to kmeans wcss.

because wcss may leave some nuance of clustering to question, we then compared a few algorithms against our twrp method, as well as our previous RPHash formulation for increasingly noisy generated data.
in this chart we quickly see the dominance of TWRP, with standard RPHash and wards method agglomerative in second.

while show superiority in this particular clustering task bodes well for the comparability of RPHash to other clustering algorithms, our main occupation is speed.
here we give a scalability plot for the RPHash algorithms, kmeans and SOTA. unfortunately the agglomerative methods are so much more computationally restrictive that they skew the plot such that there is no visible differences between the aforementioned methods. from the chart we can see that the RPHash methods beat the others hands down as the vector dimensionality grows.
in the subsequent plot we show a similar result for streamingrphash. however streaming kmeans does best it slightly.

in the dissertation we derive the following theoretical complexities. in each nm is the data set size. for RPHash it is simply linear in both.
for streaming it is linear in the input, and dominated by the dimensionality of a vector, in space.
twrp trades some computational superiority, for better clustering than standard RPHash and achieves a bound closer to streaming rphash.
epsilon and delta are related to the count error, and depth factors fromthe count min sketch.

we next investigate the potential for parallelism in rphash. while standard RPHash and TWRP show excellent performance in regard to scaled speedup, streaming rphash seems to level out pretty quickly.
this however is likely due to the effect of non-uniform memory access in multi core systems.
TWRP and standard RPHash both were presented partitioned datasets at the start, while streaming was presented vectors in a stream fashion and simply assigned tasks to worker threads, resulting in a completely random memory access pattern.
in a truly distributed system, it is less likely that this effect would dominate the speedup

now we briefly give our security assessment.
we follow a fomulation known as k anonymity, and assert that the only leaked piece of information in a distributed system would be the projection matrix.
an attacker would benefit from finding the psuedo inverse of this matrix and inverting the projection step for a shared vector.
however here we give real world assessment based on mimic 2 EKG data
Reassociating vectors using nearest neighbor after projection inversion at various projection depths. as can be ascertained from the plots, lower dimensional projections such as the 24 dimensions we generally project to, have a very low probability for de-anonymization.

this concludes our description and experimentation of RPHash algorithms.
from it we see that our TWRP algorithm and both streaming and standard RPHash are comparable to other clustering algorithms.
our hypothesis that approximate clustering vs local minima clustering holds for many real world and synthetic datasets.
This is namely due to the use of the LSH function and random projection. we have also shown that our algorithms have linear complexity, and memory bound and in the streaming case sublinear memory bound.

With what time we have left I'd like to just point to some possible future research.
first and foremost we assert that the Count-Min Cut tree deserves further investigation for problems involving space partitioning in which approximations are valid for finding solutions.
We would like to site work at UC into topological data analysis and assert the possibility of using RPHash as a dense region identifier.

this concludes the presentations, i would now like to open the floor to questions. thank you.