1. Ist ein Konstrukt aus Algorithmen die aus einem Datenbestand lernen und daraus intelligente Aktionen generieren können.

Am stärksten hängt Machine Learning von den Daten ab, je mehr aussagekräftigere Daten desto besser das Modell.

Problemstellungen: Erkennung von Spam Mails (Spamfilter), Spracherkennungssysteme, Erkennung von Kreditkartenbetrug

1. Der MAPE Loop in Bezug auf Adaption ist ein Verfahren, um ein System zu schaffen, welches sich selbst organisiert und dynamisch laufend an Änderungen anpasst.

**M**onitor – Sammeln von Daten der zu organisierenden Ressource (Topologie Infos, Metriken,…), sucht nach Symptomen welche genauer analysiert werden müssen

**A**nalyze – Führt Datenanalysen aufgrund der durch die Monitoringphase entdeckten Symptome durch. Falls Änderungen nötig sind, wird eine Anfrage an die Planungsphase weitergeleitet

**P**lan – erstellt einen Plan mit korrigierenden Maßnahmen

**E**xecute – Führt die Änderungen des Plans durch

Durch den MAPE loop können adaptive Systeme angepasst/geändert werden, ohne dass diese neu gestartet werden müssen.

1. Ist eine spezielle Art eines Rechennetzwerkes bei welchen alle Rechner gleich berechtigt sind. Alle Peers sind in diesem System gleichberechtigt. Einfache P2P Netzwerke organisieren sich selbst, das heißt es gibt keinen zentralen Server. Vorteile von P2P Systemen gegenüber Client-Server Systemen sind: es gibt keinen single Point of failure, P2P Netzwerke sind in der Regel relativ Robust
2. **Vorteil:** geringfügige Kosten 🡪 Rechner kann sofort ins Netzwerk integriert werden. Es fallen höchstens Kosten für Kabeln und Instandhaltung an. Im Gegensatz zu Client-Server Systemen bei welchen bei entsprechend großen Systemen ein teurer Server angeschafft werden muss. Je nach Größe des Netzwerkes können die Kosten varieren.

**Nachteil:** kaum Sicherheit und schwer administrierbar 🡪 Da jeder Rechner im System gleich berechtigt ist und es keine zentrale Anlaufstelle gibt können keine komplexen Berechtigungs-Hierarchien aufgebaut werden. Bei Client-Server Systemen werden alle Zugriffe vom Server verwaltet. Diese Zugriffe können von ihm geregelt werden.

1. Ein Schlüssel wird auf einen Wert gemappt. Ziel ist es Struktur zur Kommunikation einzelner Nodes in einem P2P Netzwerk einzuführen. Ein System ohne Distributed Hash Table ist bei der Suche nach einer Datei oder einem Node sehr ineffizient. Das System wird hier gefloodet, d.h. jeder Rechner schickt seinen unmittelbaren Nachbarn eine Anfrage ob er der gesuchte Rechner ist bzw. die gesuchte Datei besitzt.

**Vorteil P2P ohne DHT:** Kein Aufwand für Verwaltung der Informationen (🡪 Keine Routing- bzw. Fingertable vorhanden)

**Nachteil P2P ohne DHT:** Bei Suchen nach Rechnern oder Dateien in großen Netzten sehr inneffizient.

1. Peers sind in Chord Ringförmig angeordnet.

Jeder Peer kennt seinen direkten Nachfolger als auch

seinen direkten Vorgänger

**Ein Peer speichert folgende Informationen:** Vorgänger, Nachfolger, Fingertable

Die Navigation erfolgt über einen Index in einem virtuellen Netzwerk 🡪 Nicht über die IP

Damit die Navigation schneller vonstattengeht als ohne (Laufzeitverhalten *O(n)*), benötigt man die Fingertabelle, dazu speichert jeder Knoten weitere Nachfolger in dieser Tabelle.



**Funktionen von Chord:**

***Successor:*** Funktion lautet *find\_successor(k)* und gibt den nächsten direkten Nachfolger vom

Knoten *k* an.

***Hash Funktion:*** Um aus einen Key einen identifier zu erzeugen.

***Insert:*** Einfügen eines Key-Value-pair auf einem Knoten.

***Lookup:*** Anfragender Knoten sendet seine IP und seine Key an seinen Nachfolger. Diese Nachricht wird solange weitergereicht bis die Ressource gefunden ist. Der Knoten welche die Resource schickt diese dann direkt zum Anfragenden Knoten.

***Join the swarm:*** Ein neuer Knoten wird ins Netzwerk eingefügt.

1. Es wird ein Key-Value Pair (torrent, myIP) an einem index im Ring eingefügt.
2. Key = *hash(torrent)* 🡪 Erzeugt den Index
3. Frage den *successor(key) von*
4. Finde den zuständigen Knoten der für den key der zuständig ist für das Speichern: *successor(key)*

So wird der Index zufällig im Netzwerk verteilt. Der Key wird immer auf dem Knoten gespeichert der dem Key am nächsten ist.



1. **Join:**
2. der einzufügenden Knoten hasht seine IP-Addresse und bekommt als Ergebnis einen Key 🡪 *Key = hash(torrent)*
3. Der neue eingefügte Knoten fragt seinen Nachfolger

🡪 *successor(key)*

wer ist mein Vorgänger.

1. Der eingefügte Knoten speichert sich danach diese Information.
2. Der successor Knoten wird nun benutzt um die Fingertable des eingefügten Knoten upzudaten.

In Chord können Knoten eine DHT entweder *Graceful* (gibt seine Keys an seinen Successor weiter) oder via *Crashing* (Knoten ist einfach weg) verlassen.

1. Ist ein Routing Algorithmus basierend auf den Bellman-Ford algorithmus. Es wird versucht den kürzesten Weg in einem Graphen (Netzwerk) zu finden. Jeder Knoten hat eine Tabelle mit dem besten und günstigsten Weg zu seinen Nachbar Knoten. Diese Tabellen werden upgedatet durch Informationsaustausch und Abgleich mit Nachbarknoten. Jeder Knoten ist so in der Lage die beste Verbindung zu jedem Ziel zu berechnen.

Pro Node müssen folgende Infos gespeichert werden: *next hop, number of hops* and *sequence number*

1. Durch Link-State Routing wird der kürzeste Weg in einem Netzwerk von A nach B errechnet. In Link-State Routing werden 5 folgende Schritten pro Knoten angewandt:
2. Erforsche Nachbarn und lerne Ihre Adressen
3. Erstelle Distanz oder Kosten-Metrik zu jedem Nachbar Knoten
4. Erstelle ein Paket mit allen gerade gelernten Informationen.
5. Sende das Packet an alle Nachbarn des Knotens und empfange auch deren Informationen.
6. Berechne den kürzesten Pfad zu jedem anderen Knoten.

Berechnete Routen sind immer optimal weil der Dijkstra Algorithmus bei der Berechnung des kürzesten Pfades eingesetzt wird. Durch diesen wird immer der kürzeste Pfad errechnet.